自分のペースでゆったり学ぶ

C プログラミング 超入門

杉浦英樹



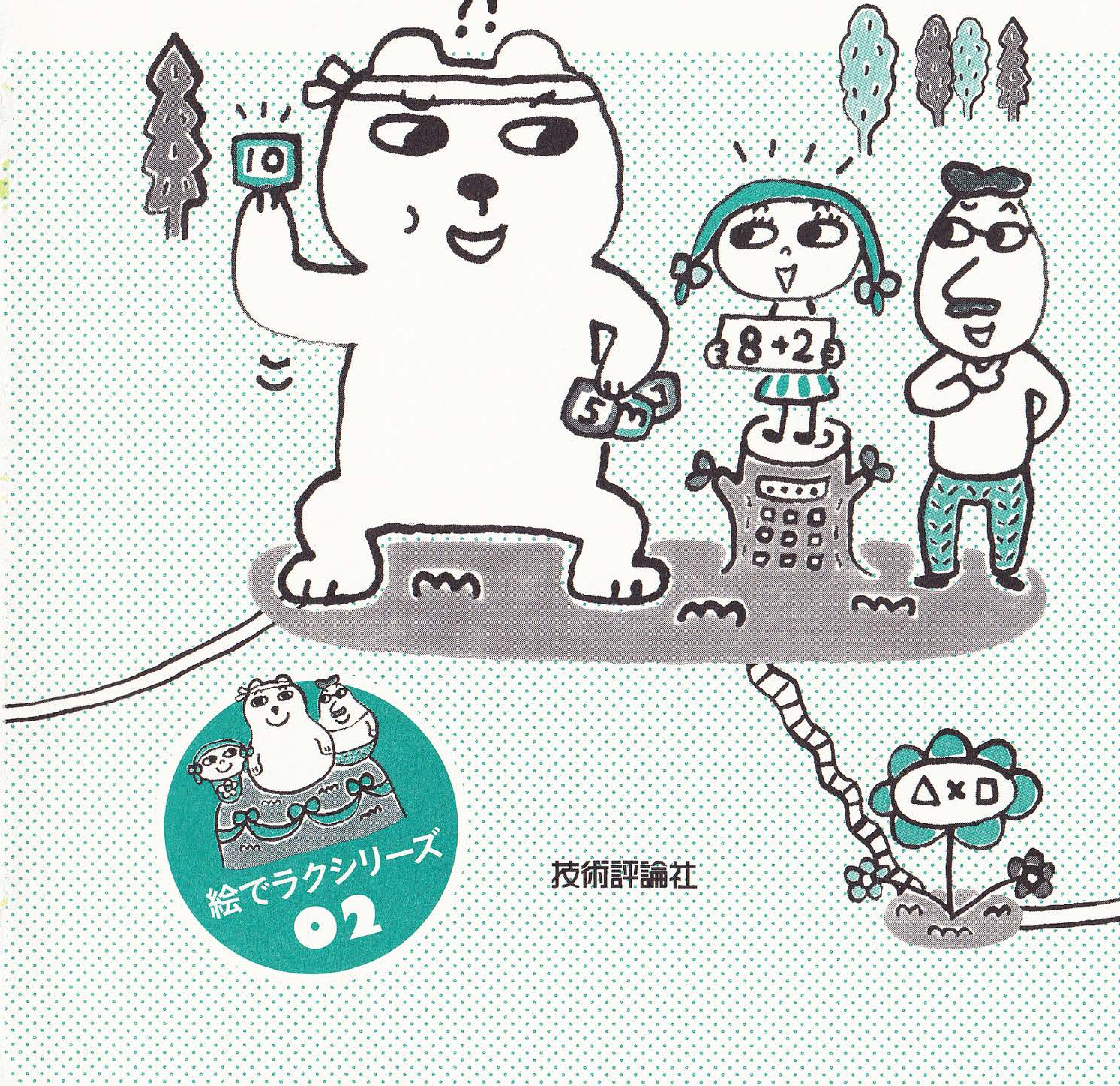
技術評論社


START

自分のペースでゆったり学ぶ

C プログラミング 超入門

杉浦英樹



技術評論社

はじめに

プログラムというと、難しい参考書が多く、敬遠していた人も多いのではないでしょうか。実は、プログラムは、コンピュータに人の仕事を手伝ってもらうために使う言葉ということができます。コンピュータは、その言葉通りに動く素直な機械です。ですから、間違った言葉をかければ、間違った動きをするのです。

この本では、人の仕事を手伝ってくれるコンピュータと、コンピュータを動かすためのプログラムの基本について、知識のない人にも理解できるようになるべく“簡単に”解説しています。

プログラミングって、どんなもの？　を知りたい人には、基本的な概念を知るために。昔、プログラミングの勉強をして、わからなくなってしまった人には、再チャレンジのきっかけとして。プログラミングの限界を感じているプログラマには、コーヒーブレークのおつまみとして、本書を利用していただければと思います。

また、近年、ソフトウェアエンジニアの不足が懸念されて久しいのですが、相変わらず、この状況は変わっていないようです。本書を手にした一人でも多くの読者から、新たなソフトウェアエンジニアが誕生すれば、筆者の本望です。

プログラムは、難しくありません。

さあ、主人公のくまと一緒に、コンピュータとプログラミングの世界を楽しんでください。

2006年4月　杉浦英樹

ことのはじまり

新しい家に引っ越ししてきた、先生と娘の絵美。
引っ越し先は、大きな森のすぐそばで、以前住んでいた場所よりは不便になったものの、自然も豊富で環境としては申し分ない所だった。

絵美「家も広くて、庭も大きい。休みの日は、お父さんと一緒に山や湖に出かけて遊べるね。」
先生「そうだね。絵美の小学校もここから見えるよ。前より少し遠くなってしまったけれど、毎日頑張って通えるよね？」
絵美「うん！」

楽しいひとときを過ごしていると、そこに何やら大きな物体が目の前に現れた。

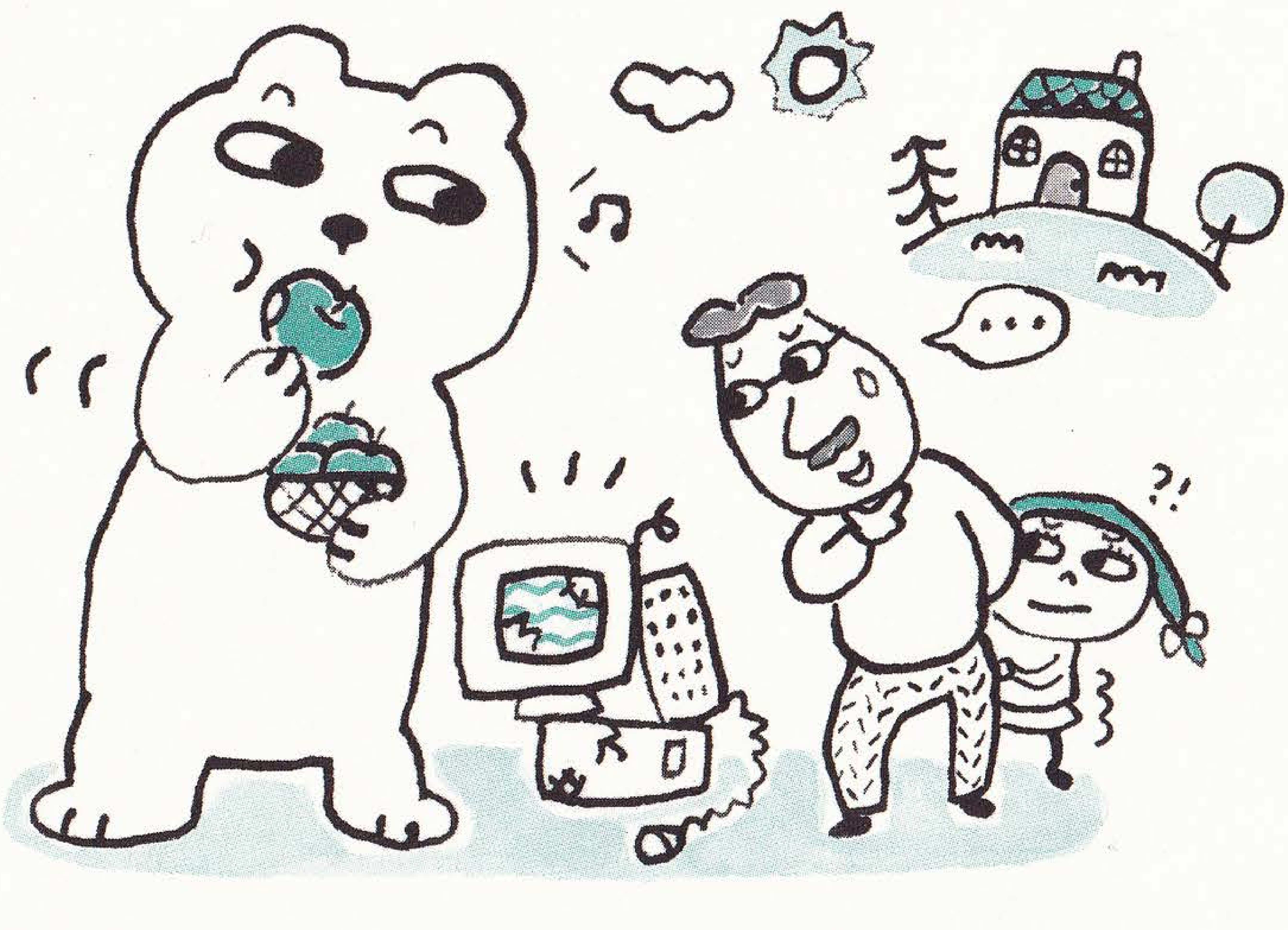
絵美「キャーッ。く、く、くま！」
先生「絵、絵美、ゆっくり、ゆっくりこっちへおいで。」

どんどんくまが近づいてくる…。

くま「あ、あのう～。これ、さっき、道端で拾ったんですけど、食べられますかね？」

何やら話ができるようだ……。

先生「そ、それは食べ物ではない。お腹が空いているんだったら、
　　　このりんごをあげるから、おとなしくするんだよ。」
くま「モグモグ…。うまい、新鮮だ。
　　　…って、りんごの方ではなくて、これは何ですか？」
先生「それは、パソコンと呼ばれる、計算をしてくれるコンピュー
　　　タだ。」
くま「へえ～。コンピュータという代物ですか…。計算していると
　　　ころ、見てみたいです。」
絵美「私も！」
先生「それより何より、まずはこのコンピュータが使えるかどうか、
　　　確認しないと。」
絵美「…あれ、何だかこのパソコン壊れているみたい…。」



もくじ

● 本書掲載プログラムについて

本書で掲載しているプログラムは、弊社ウェブサイト（以下のURL）にてダウンロードできます。

http://www.gihyo.co.jp/books/4-7741-2785-X/support

第1章
コンピュータとプログラム



あの日以来、ちょくちょく家に遊びに来るようになったくま。
最近では、私のいない昼間に家に来て、私が帰ってきてから一緒に夕食を食べて帰るというパターンになってきている。お腹を空かせなければ、娘が危険な目に遭うこともないとは思うのだが…。

先生「娘を食べたりしたら…」
くま「食べたりしませんから。」

それにしても、くまが拾ったパソコンは、あの日以来ずっと我が家に置きっぱなしだ。
新築のきれいな家に汚いパソコンがドカンと置いてある。

先生「くま君、そろそろパソコンを持って帰ったらどうだ？」

どうやらくまは、私の家にパソコンを置いておけば、パソコンが勝手にプログラミングをして、計算をしてくれると思っているらしい。まったく他力本願なくまだ。

先生「くま君、パソコンを家に置いておいたからって、プログラミングするようにはならないよ。」
くま「えっ、どうしてですか？」
先生「それじゃあ、くま君はどうやって算数を覚えたんだい？」
くま「ぼくはお母さんに算数を習ったんだ。お母さんは森の算数名人なんだよ。」

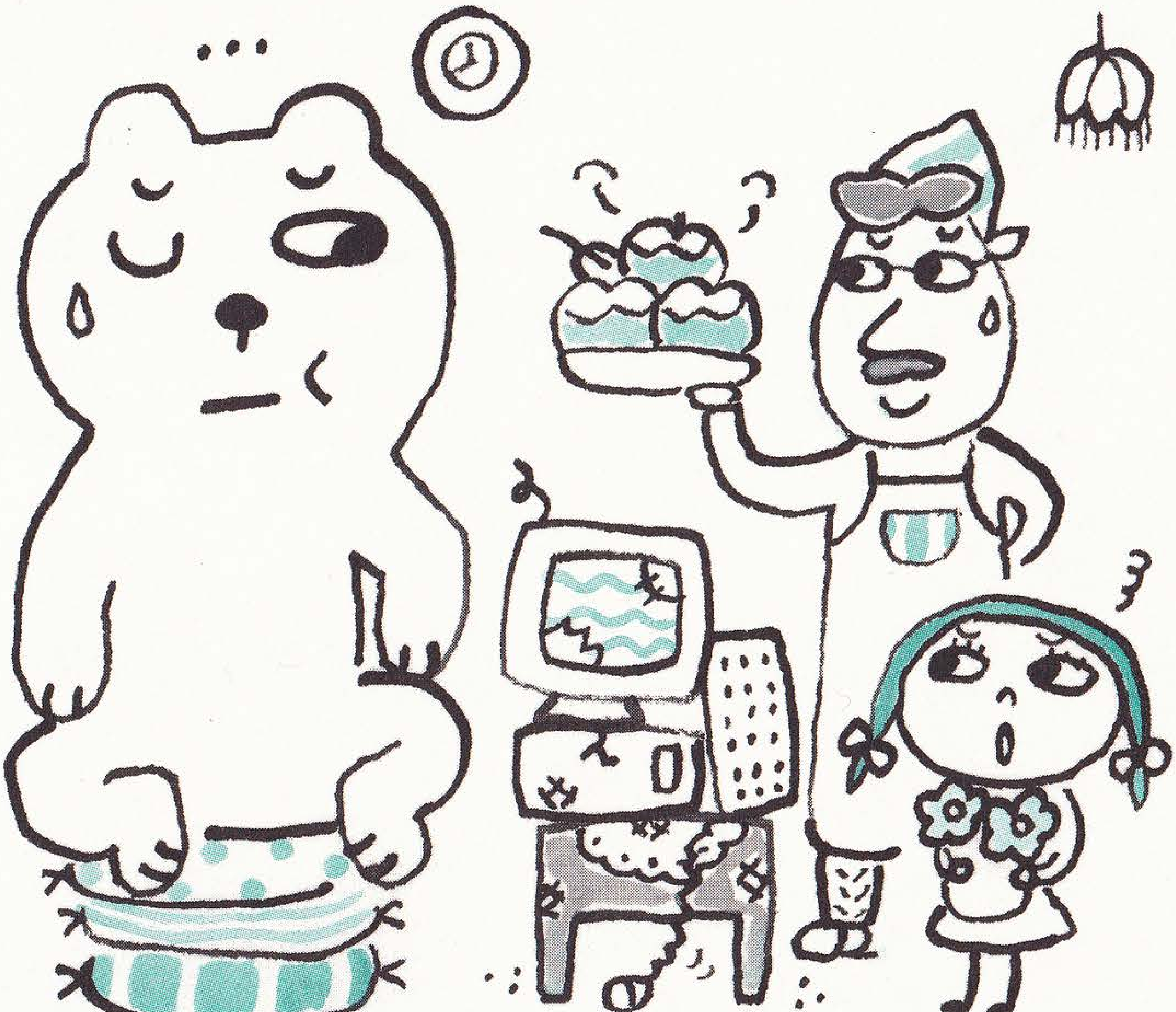
ちょっと自慢げなくま。
…一方で、くまは算数が大の苦手だった。お母さんには、森の算数選手権に出場しなさいときつくいわれてはいるが、どうもやる気がしないのだ。

先生「くま君は、算数が苦手のようだけれど、まずはお母さんに算数を教えてもらわないと計算できるようにならないように、パソコンも誰かに教えてもらわないと計算ができるようにならないんだ。」

くま「それはよくわかります。僕だってお母さんにいわれなければ、お皿洗ったり、草むしりしたりしないもの。」

絵美「わたしも！」

先生「それって、意味が違う気がするんだけれど…。」



1-1 コンピュータとは何か

● コンピュータとは

くま君のパソコンは、ボロボロに汚れてしまっている。
まったくどこから拾ってきたのだろう。家のカーペットまで汚されかねない。
絵美、汚れているパソコンをキレイに掃除してくれないかな？



「あ～面倒くさい…。あ、そうだお父さん、もう少ししたら学校でパソコンの授業が始まるんだ。」

絵美、学校でパソコンの授業が始まるっていうけれど、
そもそもパソコンって何だかわかっているかい？



「パソコン…。パソコン…。う～ん、パーソナル？　コンピュータ？」

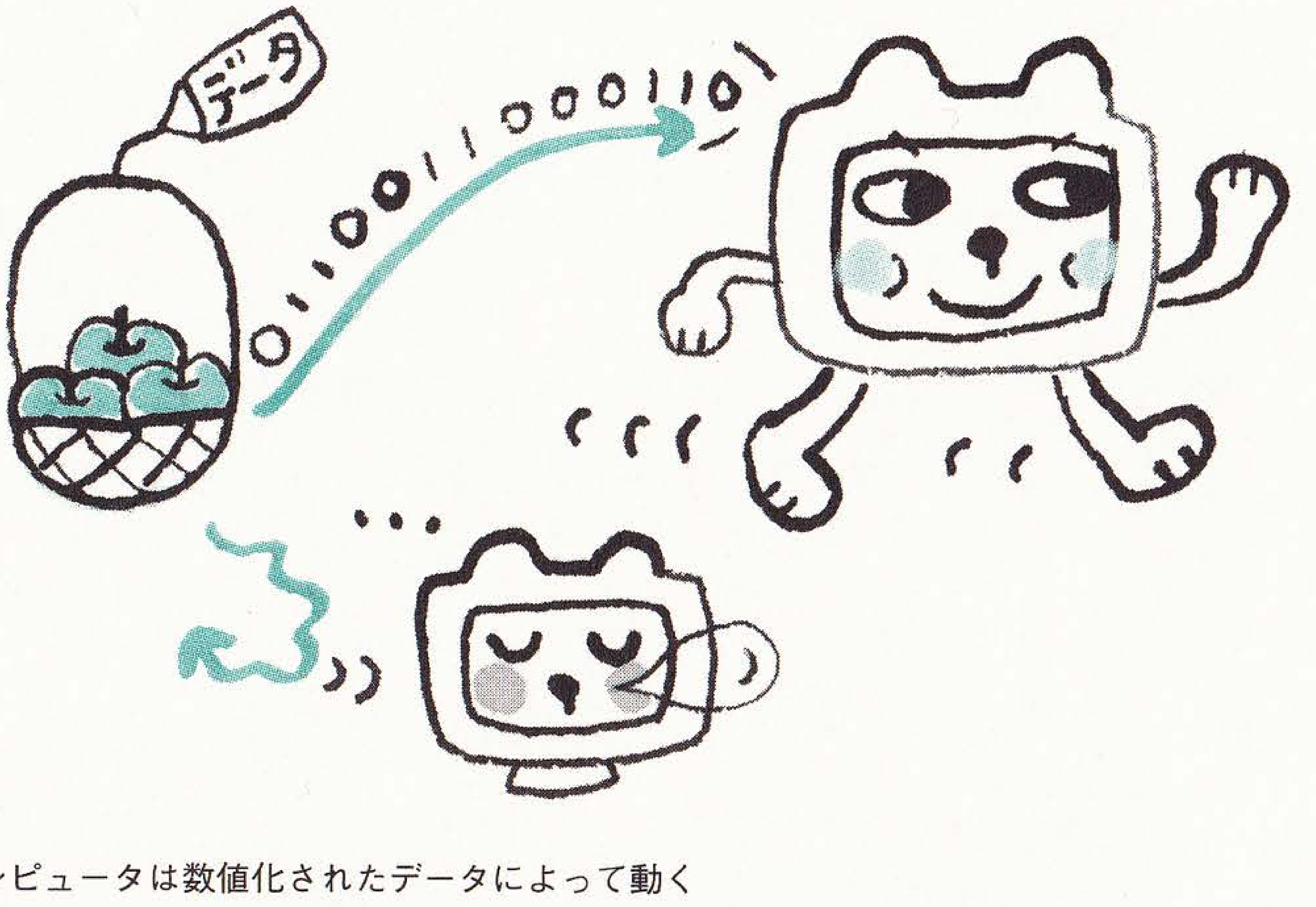
そうだ。正解。
パソコンは、「**パーソナル・コンピュータ**」を略した言葉なんだ。
パーソナルは「**個人**」という意味で、コンピュータは「**電子計算機**」のことをいう。だから、そこのテーブルの上にある電卓は「電子卓上計算機」というんだ。

「じゃあ、電卓はまさにコンピュータのことね？」

そうだね。広い意味では、確かに電卓もコンピュータになるね。
ただ、ここでいう「計算」は、「＋」「－」「×」「÷」などの足し算や掛け算の四則演算だけでなく、「**データの処理**」全般を指しているんだ。

「データ…の処理？」

ここでは、コンピュータが扱う**情報の中身の処理**のことだと思っていてくれればいい。
この情報の中身であるデータは、数字や文字、画像など様々な種類のものがある。そしてこれらのデータをうまい具合に**数値化する**ことでコンピュータは動くようになっているんだ（P41参照）。
そして、これらのデータを扱うことのできる、パーソナル、すなわち最も身近なコンピュータといえば**パソコン**になるわけだ。



● コンピュータは数値化されたデータによって動く

それに、コンピュータはパソコンだけでなく、他にも私たちの身近にある家電製品ーたとえば、掃除機や洗濯機や扇風機などの中にも入っているんだ。

コンピュータは、それぞれの機械の中に内蔵されていて、
機械を動かす仕事をしている。



● 機械はコンピュータによって動く

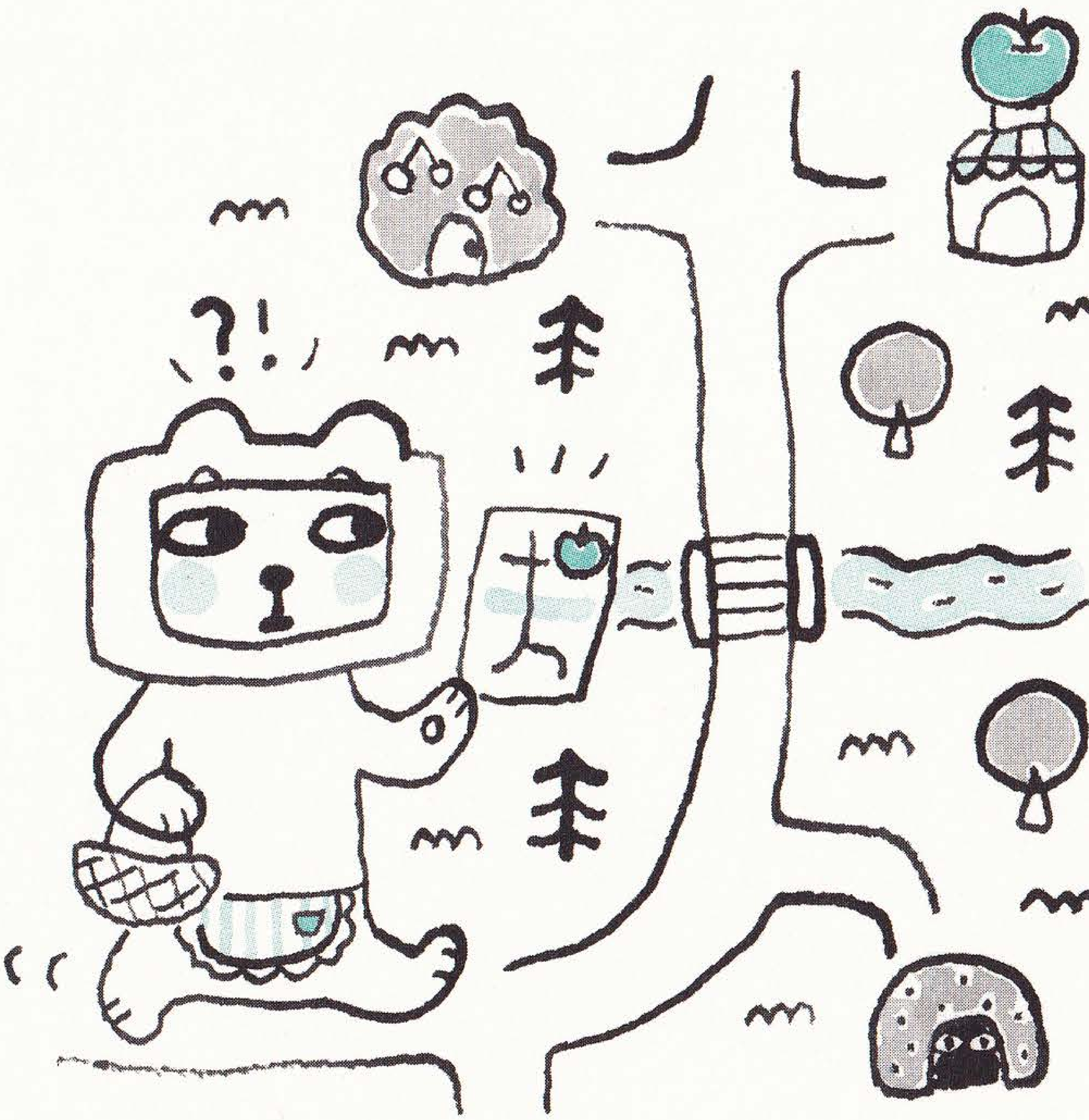
「へえ、すごいね、コンピュータって。機械の中に入れておくだけで、何でもやってくれちゃうんだ!?」

絵美、そういうわけではないんだよ。
コンピュータは、機械の中にただ入れておくだけでは一人で勝手に動くというわけではないんだ。
コンピュータは、**“誰か”からの指示がないと動けない**んだ。

くま君も絵美も、
行きたいなって思っても勝手にお買い物にいけないよね。

「うん。まずお店にどうやって行ったらいいのかわからないよ。」

そうだね。まずは、お母さんやお父さんにお店への行き方や何をどこのお店でどのくらい買うかという**指示**が書かれたメモを手渡されて、何らかの指示を受けるよね。コンピュータもそれと同じく、指示が必要なんだ。



● コンピュータも指示がないと動けない



「そうか。コンピュータは、一人では動けないんだね。誰かからの指示を待っているんだね。」

そういうことになる。コンピュータは、誰かが指示してくれることによって、はじめて動くことができるんだ。

ポイント

- コンピュータは、パソコンや家電製品に内蔵されていて、機械を動かす仕事をしている
- コンピュータは一人で勝手に動くことはない。誰かの指示によってはじめて動く

コンピュータとプログラムの関係

コンピュータは、誰かの指示があってはじめて動くことができる、ということがわかったね。

「うん。でも、誰がコンピュータに指示を出しているの？」

それは、**プログラム**だ。

「プ・ロ・グ・ラ・ム？」

プログラムっていうのはね、人間がコンピュータに
どのようなことをして欲しいのかということが書かれてある説明書
一つまり、指示書なんだ。



人間の命令が書いてある指示書＝プログラム

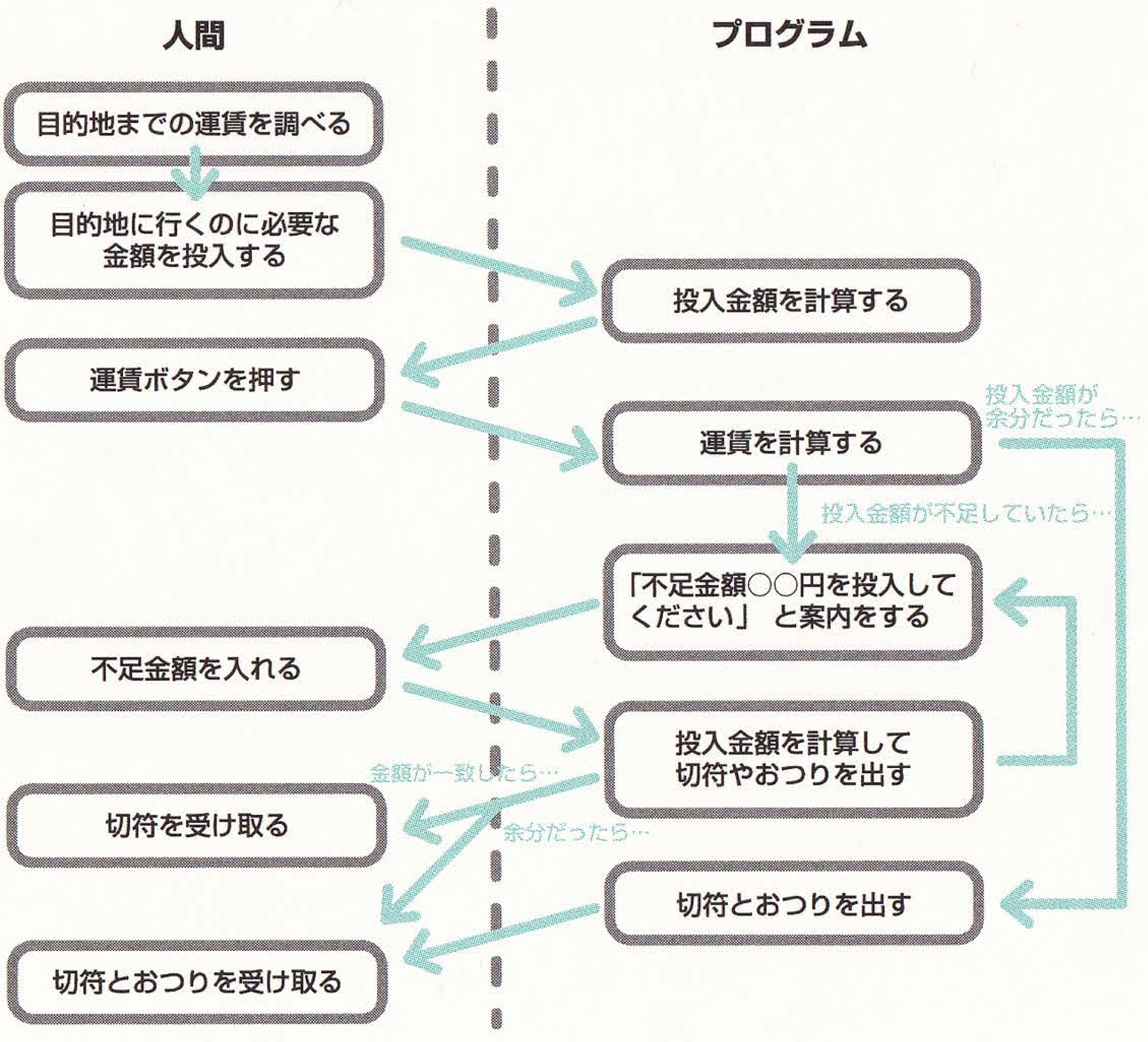
コンピュータがその説明書の決まりを守って、
その通りに動くことによってはじめて、
パソコンが使えたり、掃除機や扇風機などが動くようになるんだ。



「う～ん。でも、コンピュータの中でプログラムが動いている なんて、全然想像できないんだけれど・・・。」

そうだね。プログラムは、普通の人には見えないところで動いているからね。
私たちの身近なところにある自動券売機をみていこうか。

お金を入れて切符を手に入れるまでに
プログラムがどのようにかかわっているのか、
確認してみよう。



● 自動券売機とプログラム

このように、プログラムの処理によって、私たちははじめて切符を購入できるというわけなんだ。プログラムは目には見えないけれど、確かにパソコンの中に存在していて、コンピュータ内部で処理が行われている。

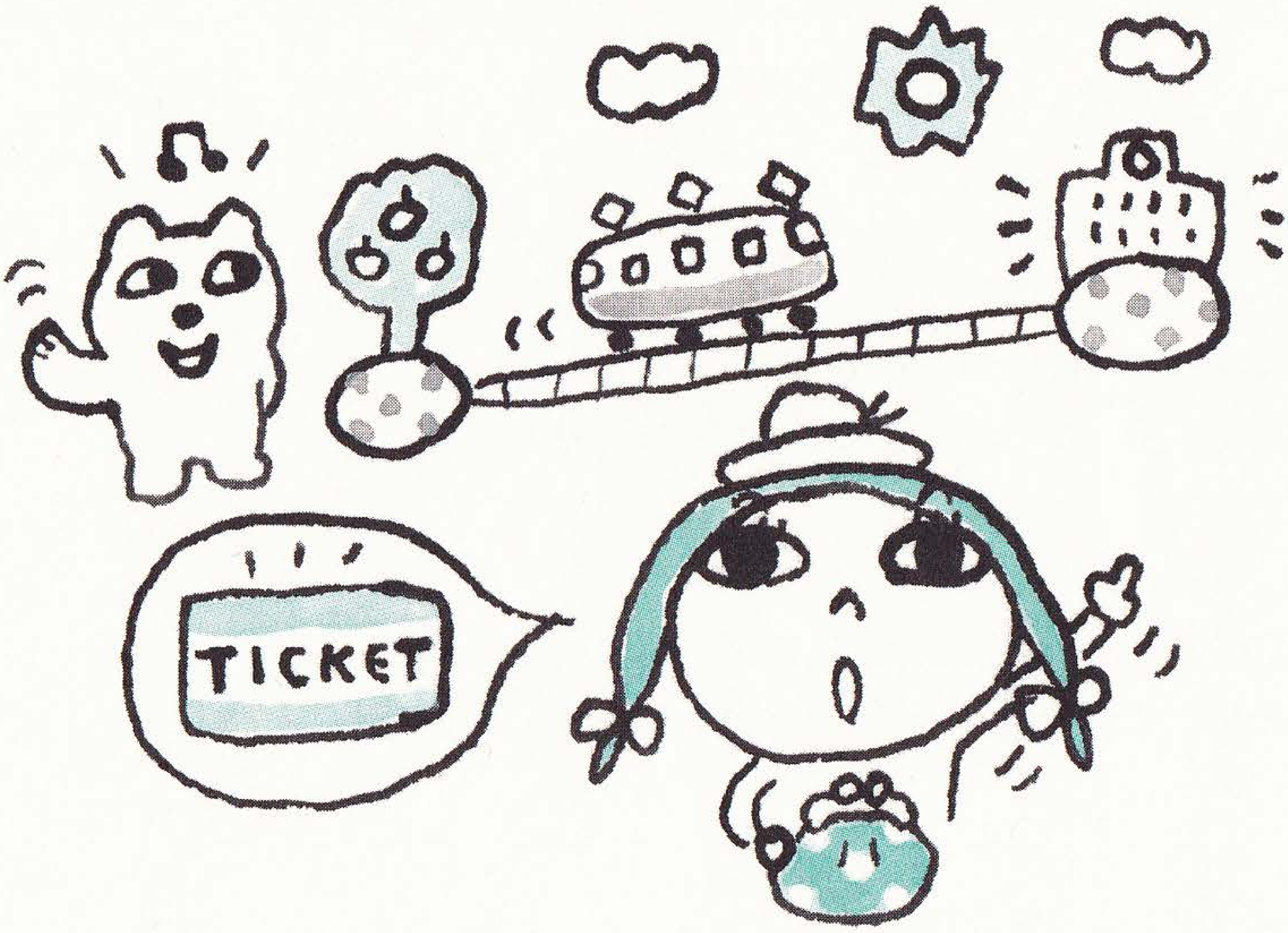
たとえばこのプログラムでは、
人間が目的地までの運賃を調べ、
運賃を投入する。そして、運賃ボタンを押すよね。

この時、投入した金額が、
不足している場合と余分な場合とで、
お客さんがそれぞれどのように対処したらいいかの案内を出す。

この時、それぞれの場合に応じて、
コンピュータは、

・どのような案内内容を出すのか
・どのようなタイミングで案内を出すのか
・その案内回数は何回か

という
細かいルールの1つひとつを順番にプログラムに書いておくんだ。
そして、これらのプログラムに書いてある指示を
コンピュータは順番に**読み取って解釈し、実行していく**んだ。
そのプログラミングの結果、私たちは、切符を手にすることができるんだ。



コンピュータはプログラムの指示通りに動く

ポイント

- コンピュータに指示を与えるのは、プログラムである
- プログラムは、人間がコンピュータにして欲しいことを伝えるための指示書である
- 人間がコンピュータにして欲しい仕事は、1つひとつ順番にプログラムにする
- コンピュータはプログラムの指示通りに動くことで、私たちは普段「パソコン」や「家電製品」などの機械を使うことができる

プログラミングとは

コンピュータとプログラムの関係は大体わかったよね。
次は、プログラミングだ。

「プログラムの-ing形（名詞化）がプログラミング？」

すごいね、いつ英語を覚えたの？
プログラミングとは、**プログラムを作成すること**をいう。
つまり、プログラムを作成するために必要な作業は、
すべて**プログラミング**として考えるんだ。

「プログラムを作成するために必要な作業？」

まずプログラミングは、プログラムの**元になる仕事**を調べるところから始まるんだ。

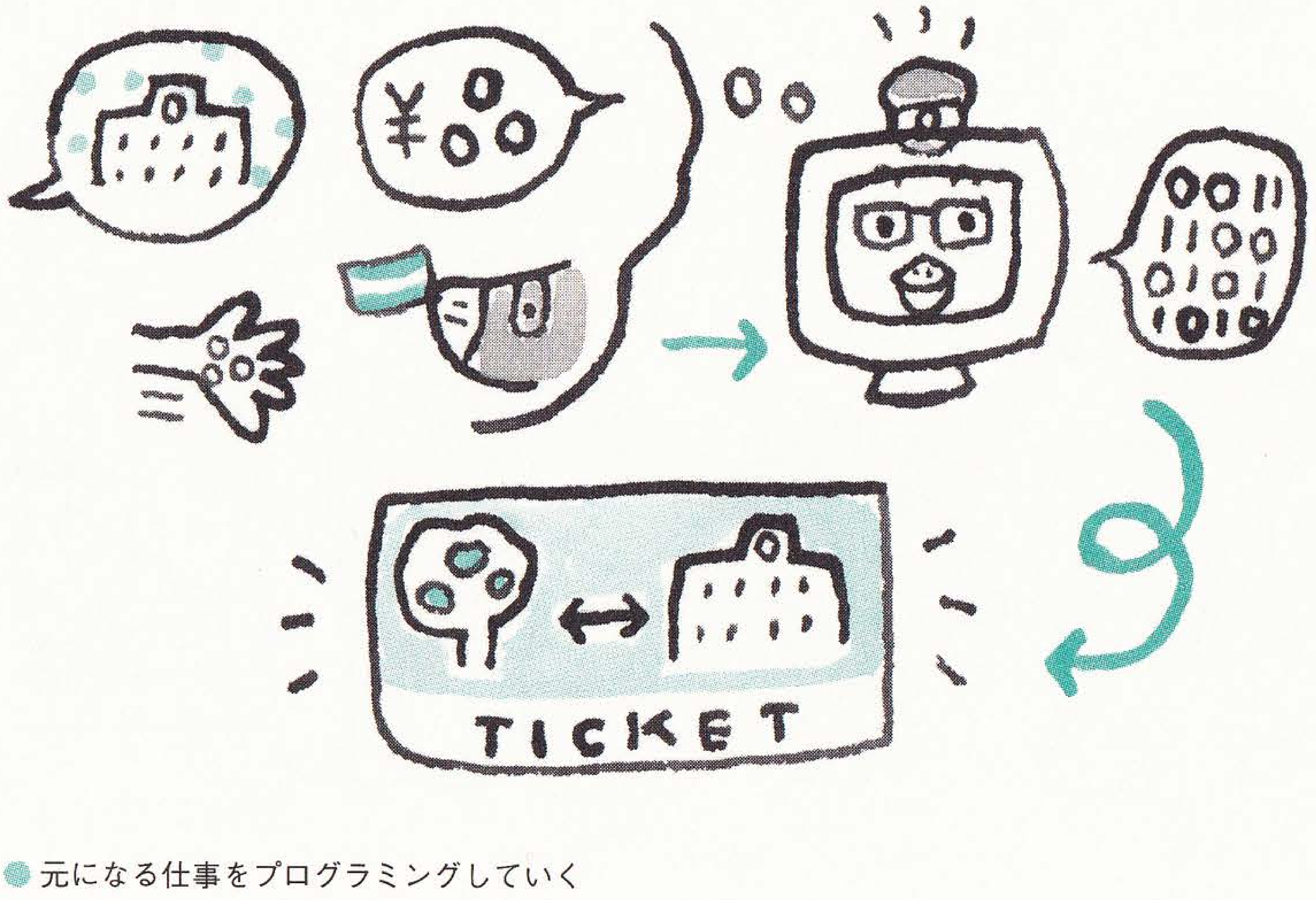


プログラミングは“プログラムの元になる仕事”を明らかにすることから

たとえば、切符の購入（P20参照）では、
目的地の駅に行くために、運賃を調べたね。
昔ならば、その後、駅員さんにお金を渡して切符を購入していた。
それが今では、自動券売機にお金を入れてボタンを押すと、
機械が目的地までの切符を出してくれるようになったんだ。

この券売機が行っている
お金を受け取り運賃を計算して、切符を渡すという、本来ならば人間が行っていた作業が「プログラムの元になる仕事」なんだ。

そして、プログラムの元になる仕事を手順として抜き出して、
それをコンピュータが**読み取り**、**解釈して**、**実行する**ように指示書を作っていく。これをプログラミングというんだ。

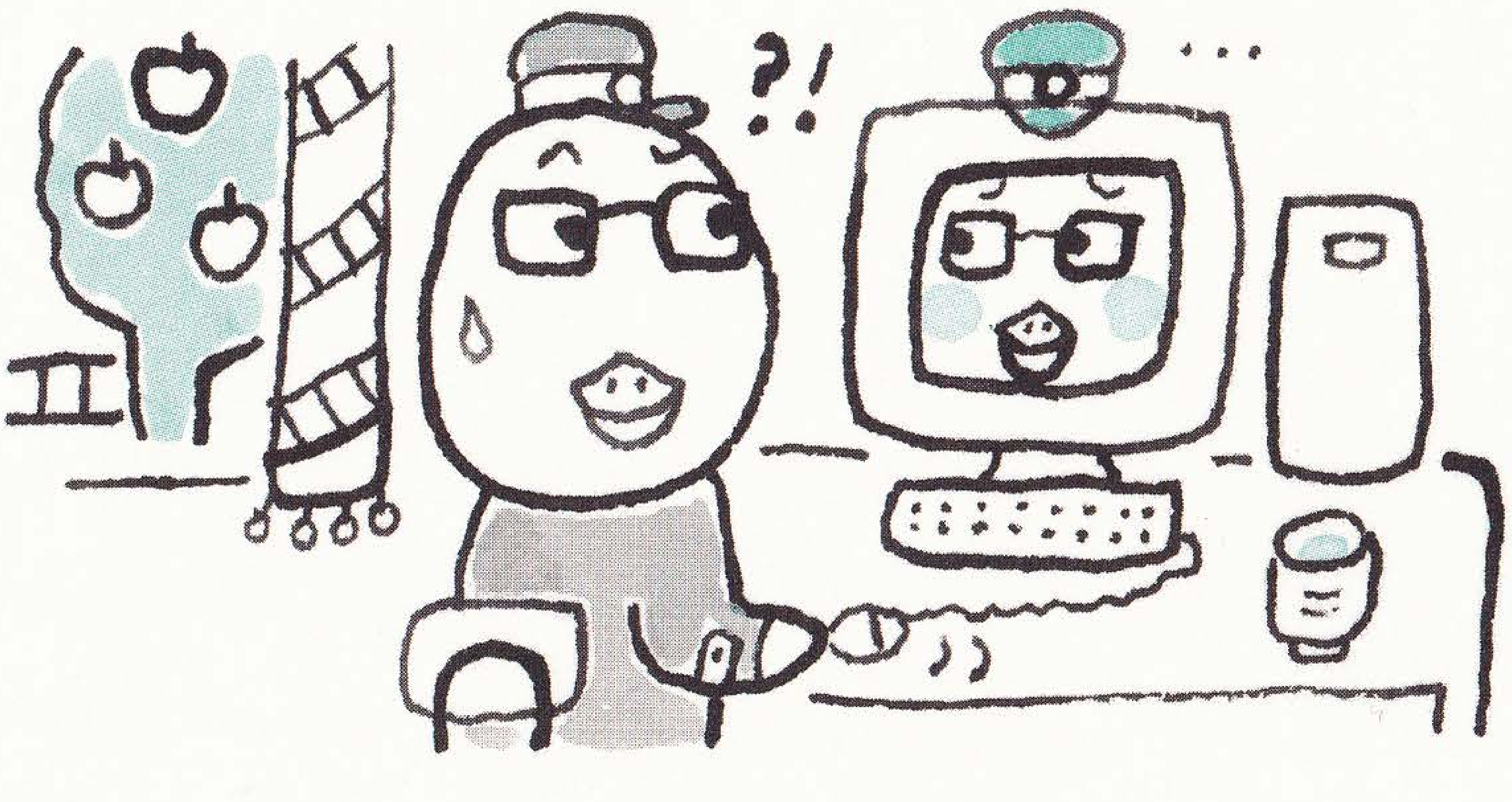


元になる仕事をプログラミングしていく

そして、プログラムが無事完成したあかつきには、
コンピュータがその**プログラムの通り動いてくれるかどうかを確認する**必要がある。実際、ここまでをプログラミングとして捉えるんだ。

もし、行きたい駅までの運賃を調べて、金額を投入したのに、
そのお金が戻ってきたり、その運賃では乗車できない金額の切符が出てきたら大変だよね。そのようなことがないように、
できあがったプログラムをテストする必要があるんだ。
このテストもプログラミングに含まれるということになる。

たとえプログラムができあがっても、
それが期待どおり動かなければ役に立たない。
だから、きちんと確認する必要があるんだ。



● テストするまでをプログラミングという

ポイント

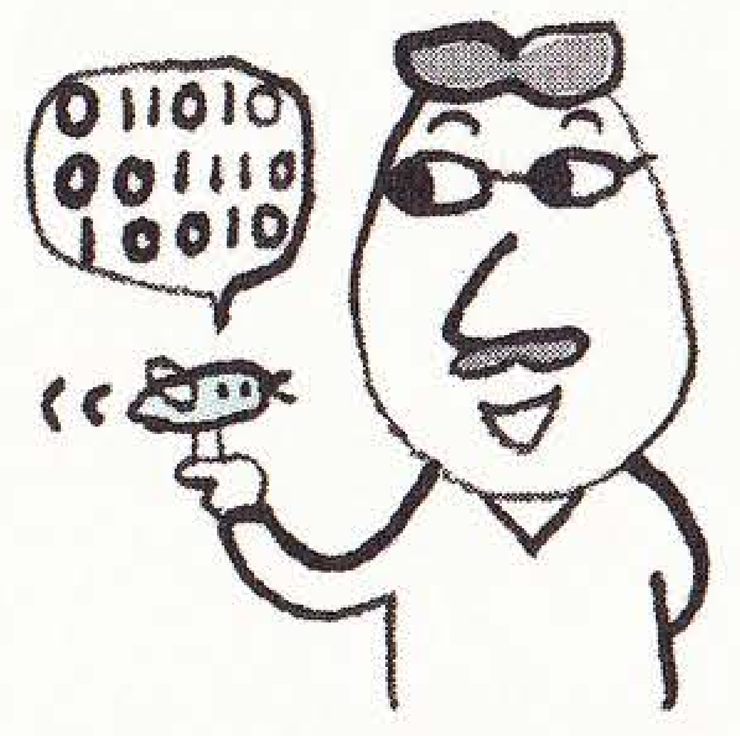
- プログラミングとは、「コンピュータにして欲しい仕事」を調べて、プログラムを作成することをいう
- プログラミングはプログラムが完成後、指示通りに動くかどうか、テストをするすべての作業をいう

まとめ

- コンピュータとは、情報の中身（データ）を扱うことのできるパーソナル・コンピュータ（電卓、パソコン、家電製品など）のことをいう
- コンピュータは、プログラムの指示によって動く
- プログラムは、人間がコンピュータにして欲しいことを伝えるための指示書である
- プログラミングは、コンピュータにして欲しい仕事を調べて、プログラムを作成することをいう
- プログラミングは、プログラム通りにきちんと動くかテストをする作業のことまでをいう

第2章

コンピュータのしくみ



くまがパソコンにりんごを詰めている。

先生「おいおい、くま君、何しているのかね。」
くま「このパソコン、このままじゃ計算できないようなので、計算ができるようにりんごを詰めているんです。」
絵美「どうしてりんごを詰めているの?」
くま「ぼくは算数を解く時にりんごを使って計算をするんだ。パソコンもこのままの状態では計算ができないから、きっとりんごが必要に違いないんだ…。」

くま君、パソコンにりんごを詰め込んでも、何にも起こらないよ。

くま「どうしてですか?」

先生「このパソコンにはメモリとCPUが付いていない。だから、メモリとCPU以外のものを入れたとしても、計算することはできないんだ。まして、果物のりんごじゃ、どうしようもないよ…。」

くまは、何やらリュックサックをゴソゴソ探している。

絵美「くま君、まだりんごを詰めるつもり?」
くま「先生、これ、パソコンと一緒に拾ったんですけど、関係ありますか?」
先生「おお、それこそまさしくメモリとCPUだ!」
くま「もしかしてこの2つの部品さえあれば、パソコンは動くようになりますかね?」

先生「まだ他の部品を確認していないから何ともいえないけれど、これらは、パソコンが動くのに必要不可欠なものだ。」

部品が揃えば、パソコンが組み立てられるが…。
待てよ、まだりんごが残っている…。
くま君、まずは、りんごを出してくれ。

くま「あっ、はいっ、わかりました。」
絵美「くま君、私も手伝ってあげるよ。でも、なんだか、りんごの蜜でベタベタする。ちゃんと掃除しないと……」

パソコンを組み立てるには、ほど遠いようだね…。

くま「とほほ…。」



2-1 コンピュータのしくみ

コンピュータの構成

さて、パソコンからすべてのりんごを取り出すことができた。
パソコンの部品を組み立てていこう。


「組み立てって、ワクワクするなあ。」


「お父さん、さっきのメモリとCPUを持ってきたよ。」

メモリとCPUは、大切な部品だ。
他の部品も見たところ一応揃っていたから、パソコンの部品は全部揃ったことになる。
これらを正しい位置に配置して、正しく組み立てていくんだ。
つまりコンピュータが動く条件は、**必要な部品がきちんと揃っていること**、そして、**あるべきところにそれが正しく配置されていること**なんだ。

加えて、**プログラム**（P19参照）が必要になる。
プログラムがコンピュータに組み込まれていないと、
実際に人間がして欲しいことが実現できないからね。


「パソコンを動かすには、部品とプログラムが必要ということね。」

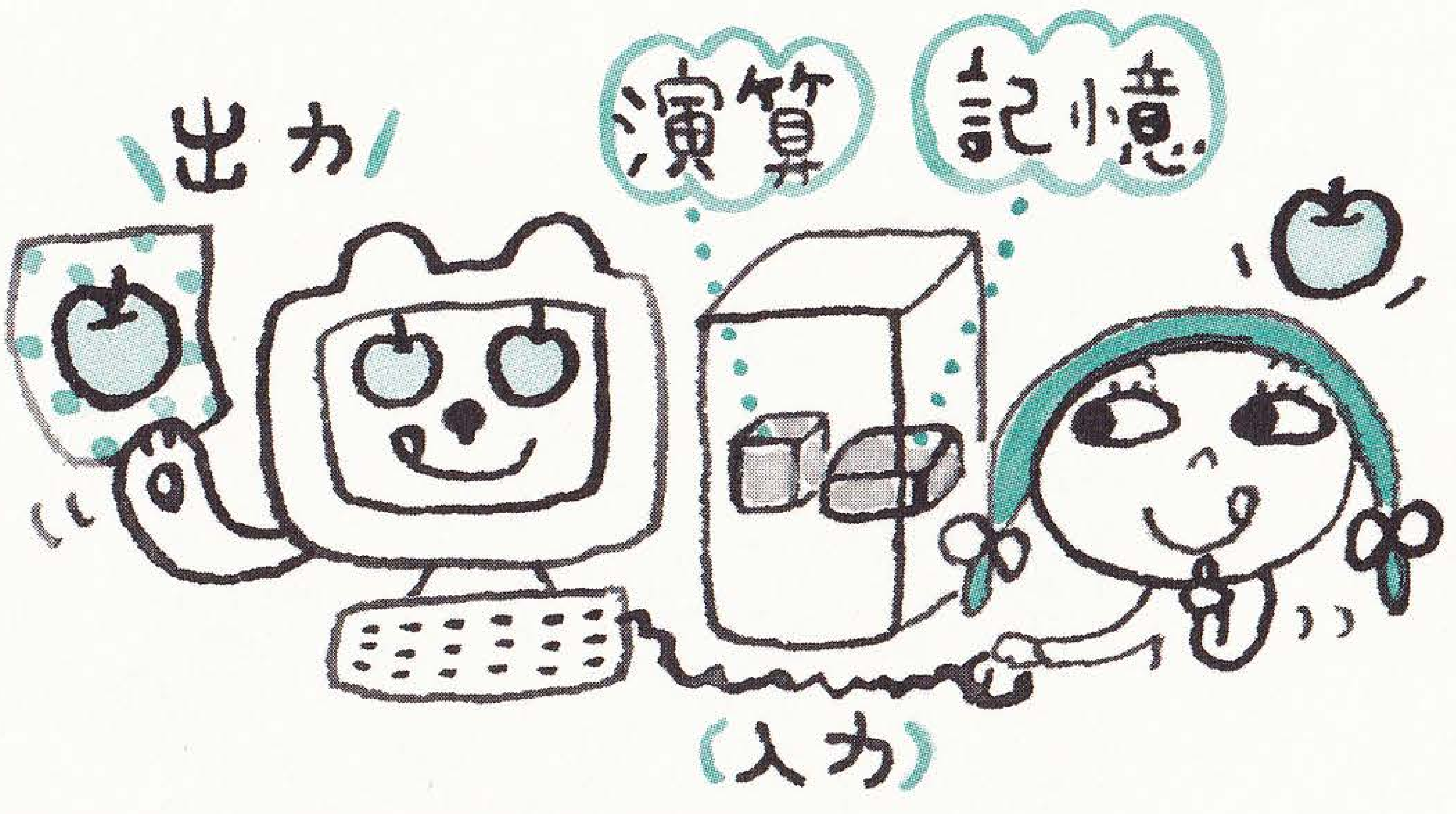
そういうことになる。
では、実際にコンピュータを構成する部品をみていこう。

部品は4つの装置に大きくわけることができる。

・入力装置
・演算装置
・記憶装置
・出力装置

「にゅうりょく、えんざん、きおく、しゅつりょくの装置?」

そうだ。
では、これら4つの装置について
1つずつみていこう。



「入力装置」「演算装置」「記憶装置」「出力装置」

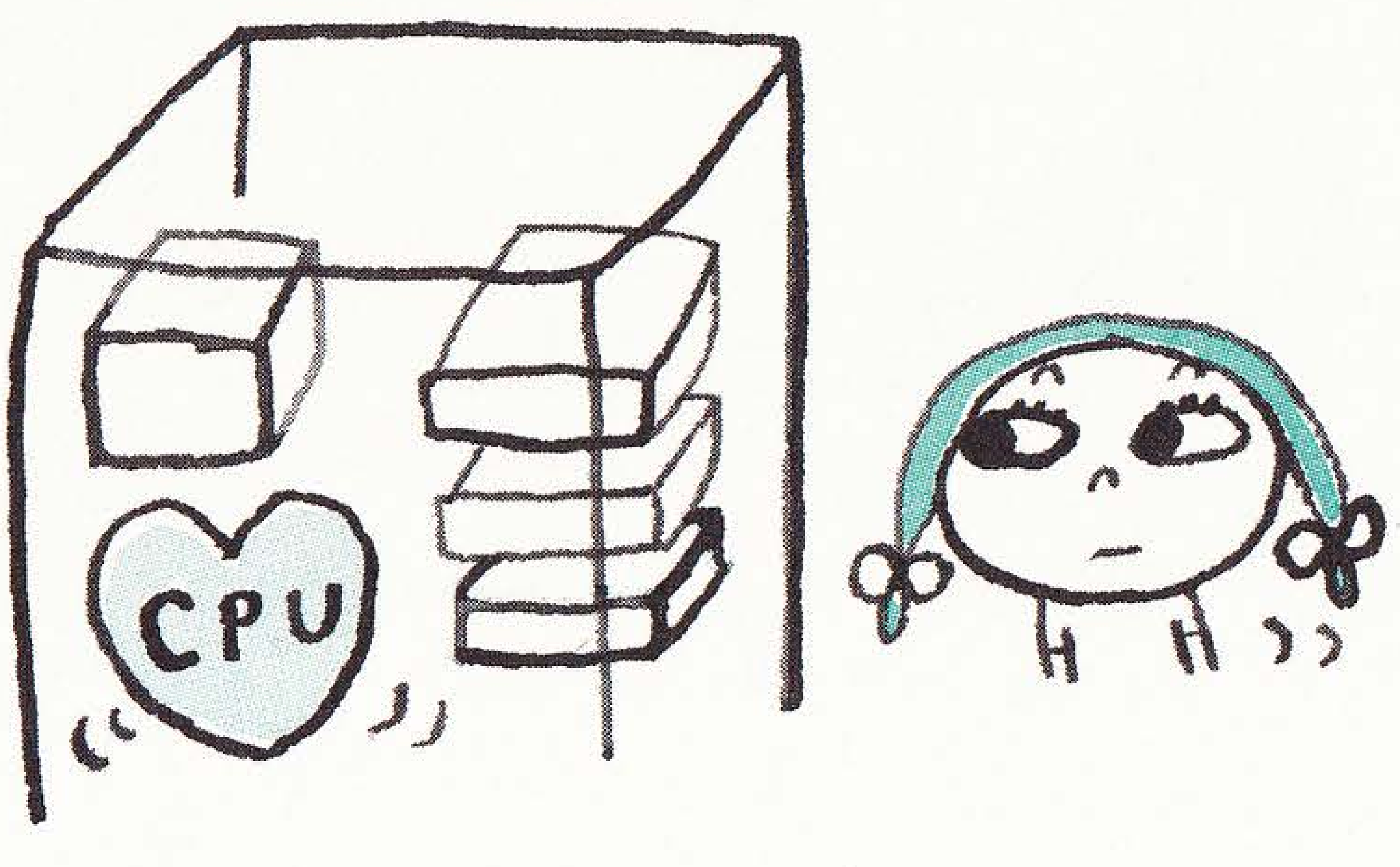
まず、**入力装置**というのは、**キーボード**や**マウス**のことだ。
人間がコンピュータに対して、**して欲しいことを直接指示するところ**だ。
人がコンピュータを動かすために「**指示を与える**」重要な部分といえる。



入力装置

次に**演算装置**だ。演算装置は**CPU**のことを指している。
キーボードやマウス等の入力装置の指示に従って、計算を行う装置のことで、**プログラムを実行する**場所になる。

CPU（シーピーユー）（Central Processing Unit）は、直訳すれば「**中央処理装置**」のことで、コンピュータ機能の中枢といえるんだ。



演算装置

そして、**記憶装置**。

記憶装置は、パソコンの**メモリ**を指している。

メモリにはプログラムが入っていて、

実際に仕事をする時に**必要な情報を読み出す**ことができるんだ。

演算装置があっても、それを働かせるための「**プログラムを記憶しておく場所**」がないとコンピュータは仕事ができないからね。

そのため、記憶装置はとても大切な部品なんだ。



● 記憶装置

最後に**出力装置**。この装置は**モニター・ディスプレイ**や**プリンタ**を指していて、コンピュータの**仕事の結果を表す**装置といえる。

モニター・ディスプレイやプリンタには、
文字や表、グラフ等の形で
コンピュータの仕事の結果が表示されるんだ。



● 出力装置

ポイント

- コンピュータは、入力装置、演算装置、記憶装置、出力装置の４つの装置から構成されている
- 入力装置は、コンピュータを動かすための命令を出す装置のことをいう
- 演算装置は、入力装置の指示をもとに計算をする装置のことをいう
- 記憶装置は、コンピュータに命令するのに必要なプログラムを記憶する装置のことをいう
- 出力装置は、一連の仕事の結果が表れる装置のことをいう

コンピュータの働き

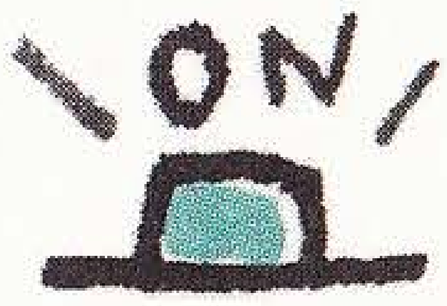
コンピュータは、
入力装置、演算装置、記憶装置、出力装置
の4つの部品から構成されていることはわかったよね。

これらすべての装置が整った状態で
人間が命令した**プログラムを組み込む**ことで、
コンピュータは、はじめて一人前の仕事ができるんだ。

パソコンに仕事をさせるには、
うまくこれらの装置を使って、**コンピュータに“して欲しいこと”** を
伝えていくんだ。
では、人間がどうやって命令を出していくのかをみていこう。

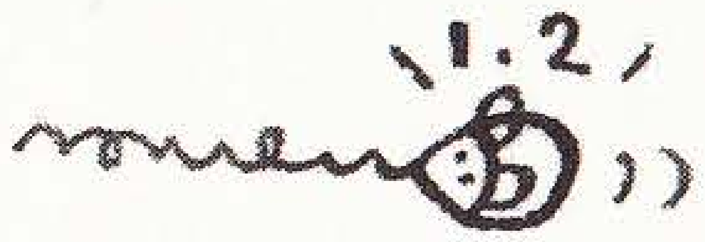
① パソコンの電源を入れる

↓



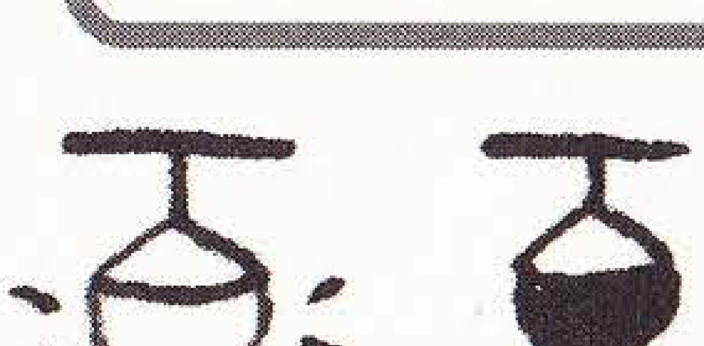
② 電源を入れた後、入力装置（マウス、キーボードなど）を使って、コンピュータにして欲しいことを指示する

↓



つまり、人間からの指示は、
入力装置を通して伝えられていくんだ

③ 入力装置からの指示が電気信号に変換される



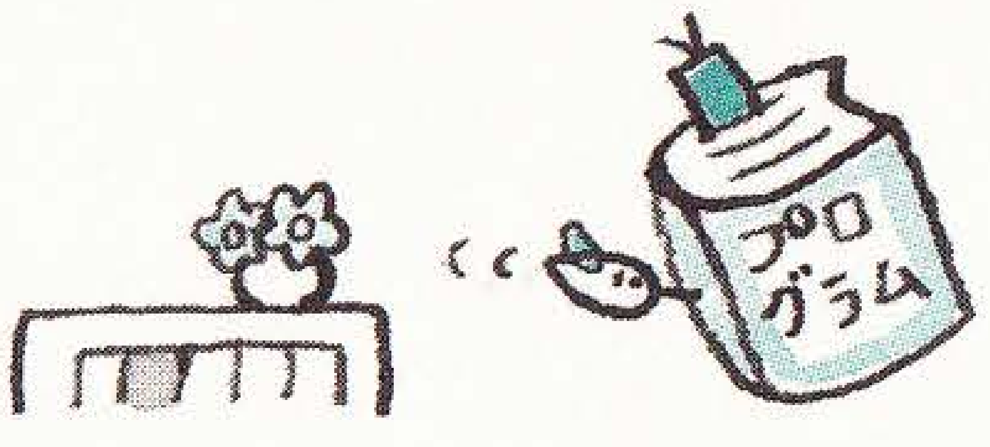
この入力装置からの指示が
電気信号に変換されることで、
コンピュータは指示の内容を理解することができる

④　電気信号が演算装置 (CPU) に伝えられる



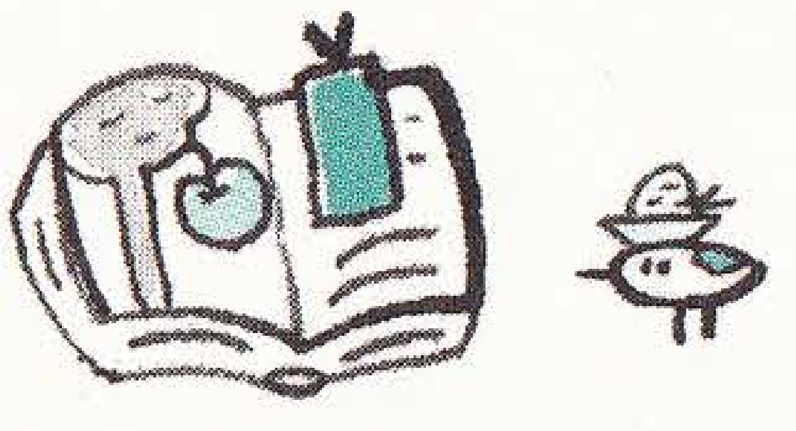
入力装置と演算装置 (CPU) は
電線でつながっていて、
電気信号に変換された指示が電線を通して伝えられる

⑤　伝えられた指示に対応するプログラムが
記憶装置から読み出される



CPU は、電線を通して伝えられた
「指示」に対応する「プログラム」を
コンピュータの記憶装置の中から読み出すことになる

⑥　演算装置は、プログラムを認識し、
プログラムの通りに実行する



CPU は、記憶装置から指示に対応した
プログラムを持ってきて、
プログラムに書いてある通りに実行するんだ

⑦　プログラムの通りに処理した結果が
出力装置に表示される

出力装置に結果が示されれば
プログラムが実行できたということだ



さて、あるプログラムを実行したときの、パソコンの処理の流れを実際にみていくことにしよう。

絵美、何かパソコンにやって欲しいことはあるかな?



「私、日記を書きたいんだけれど…。パソコンで日記を書くことってできるのかなあ?」

では“Wordを開く”命令をパソコンでやってみよう。

① **パソコンの電源を入れる**



↓

② **パソコンが起動したところで、デスクトップにある [Microsoft Office Word] のアイコンをダブルクリックする。このダブルクリックは、入力装置 (マウス) を使うことで実行できる**



↓

③ **[Microsoft Office Word] アイコンをダブルクリックすることで、パソコンに与えた命令 (Word 文書を起動させよ) が電気信号に変換され、演算装置 (CPU) に伝えられる**



↓

④ **CPU は伝えられた情報からWord 文書を起動させるという内容に対応したプログラム「Microsoft Office Word」を記憶装置から読み出す**



↓

⑤ **CPU は、対応したプログラムを認識し、プログラムの通りに処理をする**



↓

⑥ **Word 文書が出力装置のデスクトップ画面 (モニター・ディスプレイ) に表示され、「Word 文書」が開かれる**



このように人間の指示命令は、パソコン内で電気信号に変換される。
このとき**電気信号に変換された指示**は、**演算装置を通して認識され**、対応するプログラムを**記憶装置から読み出し**、**処理を行う**ことで命令を実現することができる。

このとき、絵美は**Word文書を開く**ということをコンピュータに指示することで、該当するプログラムが動き、日記を書くことができるわけなんだ。



● 日記を書く絵美

ポイント

- コンピュータは、入力装置、演算装置、記憶装置、出力装置のすべてを用い、役割を分担し仕事を実現している
- 入力装置から与えられた指示は電気信号に変換され、演算装置に伝えられる
- 演算装置は、記憶装置から対応するプログラムを呼び出す
- 演算装置は、読み出したプログラムの通りに処理を行う
- 出力装置は、プログラムを実行した結果を表示する

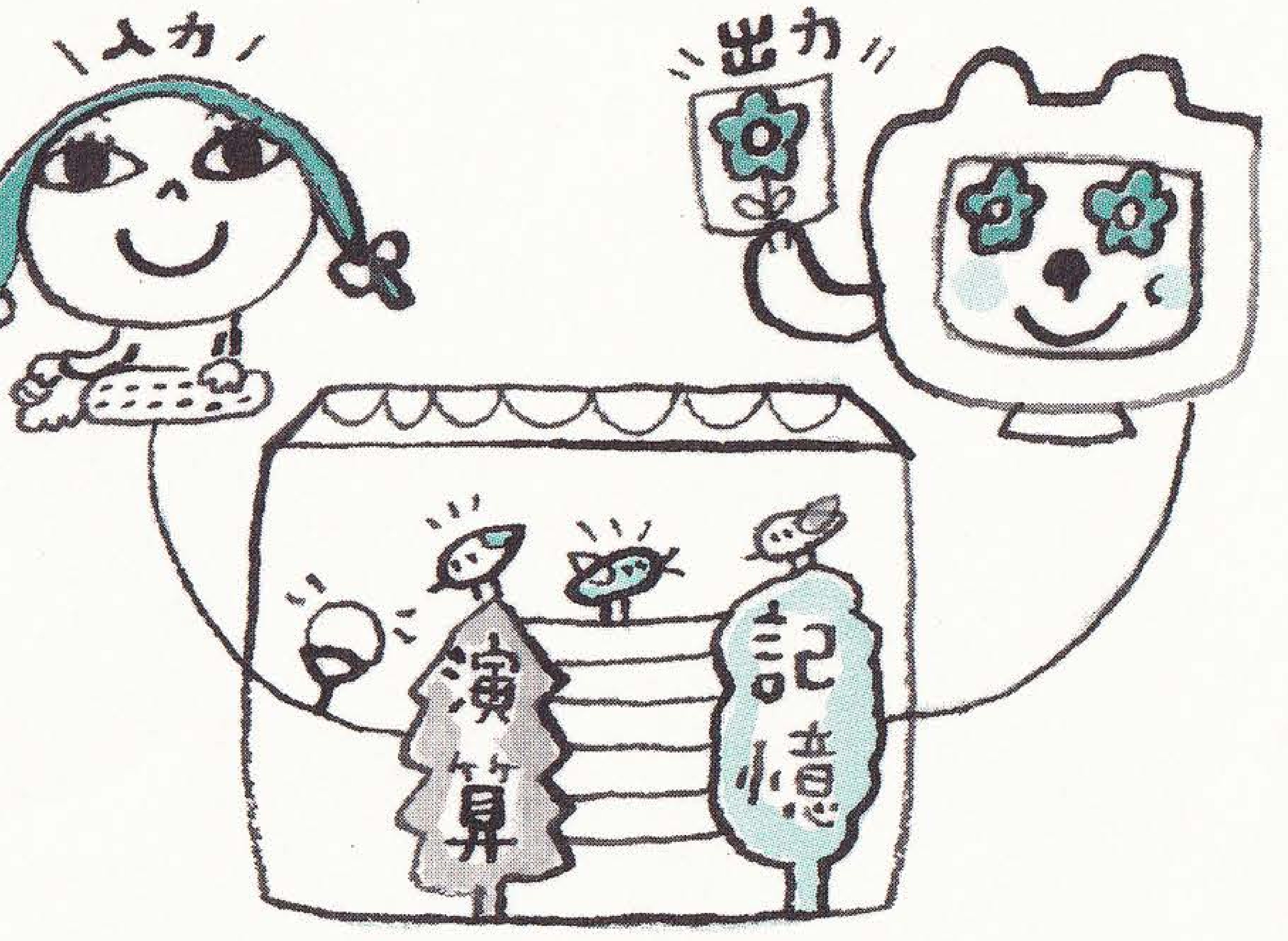


2-2 データとは何か

● データとは

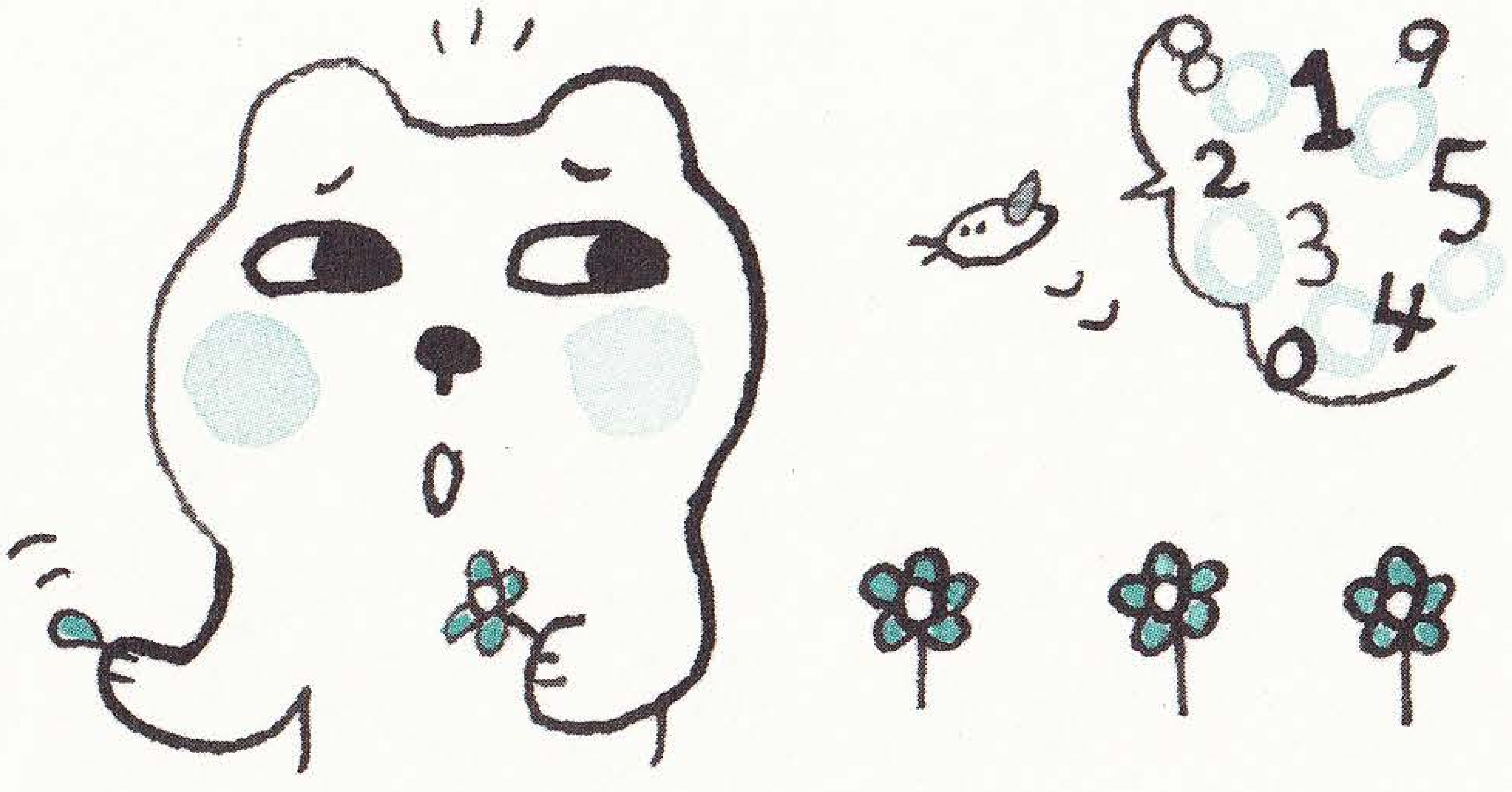
人間はコンピュータにして欲しい**何か**を**プログラムに変換**させて、
そのプログラムを実行するように指示していたね。

そして、コンピュータはその処理を
入力装置、演算装置、記憶装置、出力装置の
4つの装置を使って動かしている（P31参照）。



● コンピュータへの指示は4つの装置間で行われる

このとき各装置を通して伝達される指示は、
別名“**データ**”とも呼ばれる。
たとえば、普段私たちが見ている、画像や文書ファイルもデータなんだ。
そして、このデータ（情報）は数字でできているものなんだ。



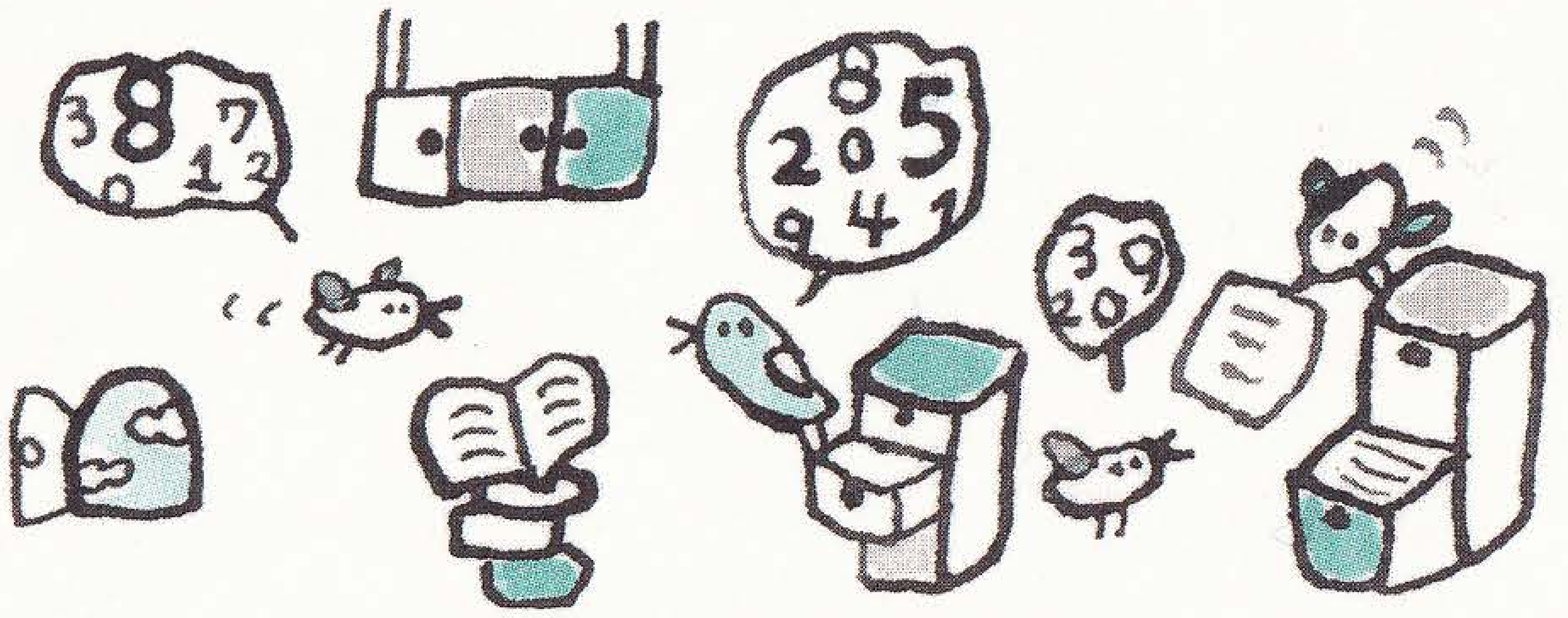
● データは数字の集まり

「どうして、指示をデータ（数字）として扱うの?」

コンピュータが動作をするためには、
コンピュータが理解できるように、必要な情報を
すべて**数字に置き換えて、データ化（数値化）する**ことが大切なんだ。

人間は、意思を伝えたいとき、日本語、英語、中国語などの
コミュニケーションの媒介となる言語というものを使ってお互いに
伝えているよね。

コンピュータも人間と同じように、
コミュニケーションの仲介となるもの（言語）が必要になる。
つまり、コンピュータが情報を理解するには、
データ（数字）を**媒介**として（P40参照）、
コンピュータ内部で意思を伝え合っているということになるわけだ。

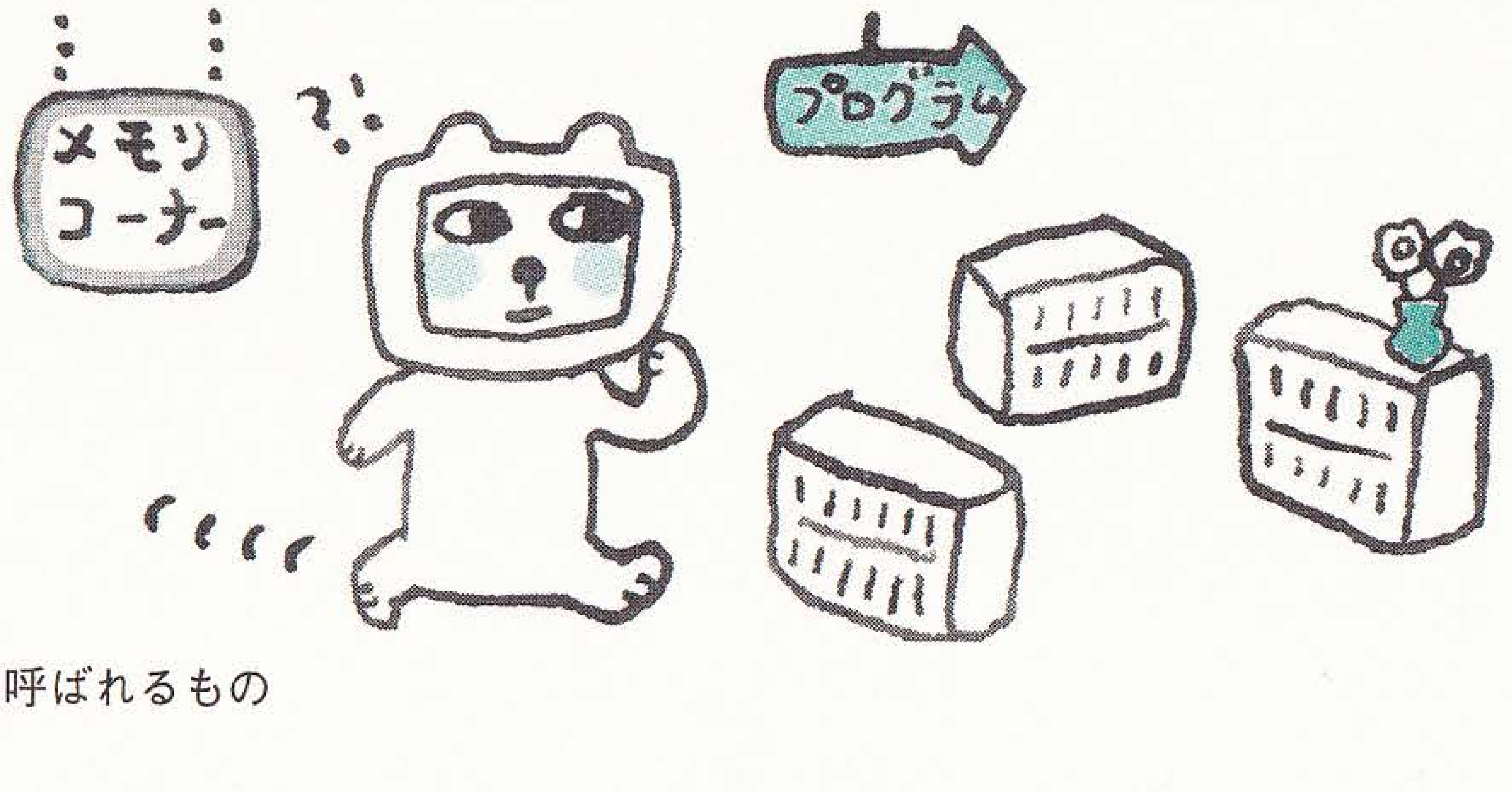


● 指示はデータとなり、コンピュータ内部で意思を伝え合う

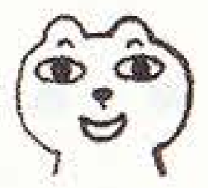
このデータと呼ばれるものは、

・**プログラム**そのもの
・メモリの**どこにプログラムがあるか**、または**どこにあるかを示すもの**
・プログラムの中で使う、足し算や掛け算などの**計算方法**

などがある。



● データと呼ばれるもの



「プログラムも、計算内容もすべてデータであり、数字でできているんですね。」

プログラムだろうと、計算途中に出てくる数字だろうと、
コンピュータが扱っているものは、
すべてデータということができる。

このデータによって、
コンピュータ間ではやりとりが行われ、
コンピュータは動くことが可能になるんだ。



「やりとりはどこで行われるの?」

演算装置である**CPU**と記憶装置である**メモリ**との間での**データのやりとりが行われる**んだよ。

記憶装置（メモリ）には、プログラムなどのデータが記憶されていて、演算装置（CPU）は、このメモリの中にあるプログラム（データ）を読み出して、仕事の内容を認識し、実行と処理を行っている。

ポイント

- 人間の指示は、すべてデータとして置き換えられる
- データは、各装置間で認識され、処理が行われる

ONとOFF

各装置へ伝えられる情報は、
すべてデータ化されることで
コンピュータは動くということが理解できたかな?

ここでは、このデータがコンピュータにどのように伝えられ、
認識されていくのかを見ていくことにしよう。

コンピュータは、家電製品と同じように電気で動いているんだけれど、
この電気信号のやりとりで使われるデータは
電球の**ONとOFF**の2種類の信号で表すことができる。
電球を使って考えてみよう。


「ONとOFFって?　どういうこと?」

電球が
ONなら「光っている」
OFFなら「消えている」

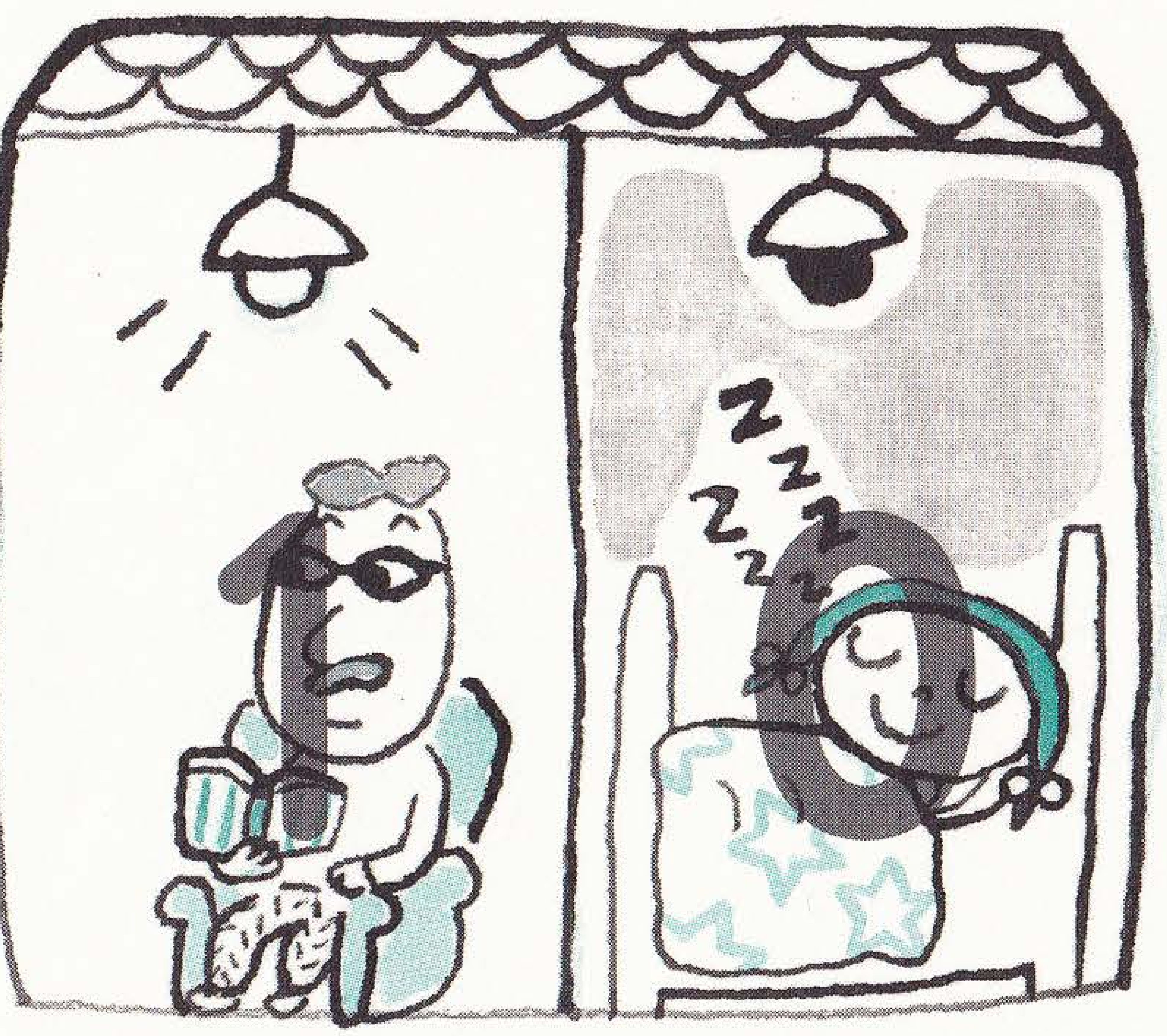


ONなら光っている、OFFなら消えている

つまり、
何もない（消えている）を「0」とすると、
何かある（光っている）を「0以外」ということができる。
0以外の何かあるを一番小さい数字の「1」として考えてみよう。

すると、
ON（光っている）の場合は「1」
OFF（消えている）の場合は「0」
の2種類の信号で表すことができるね。

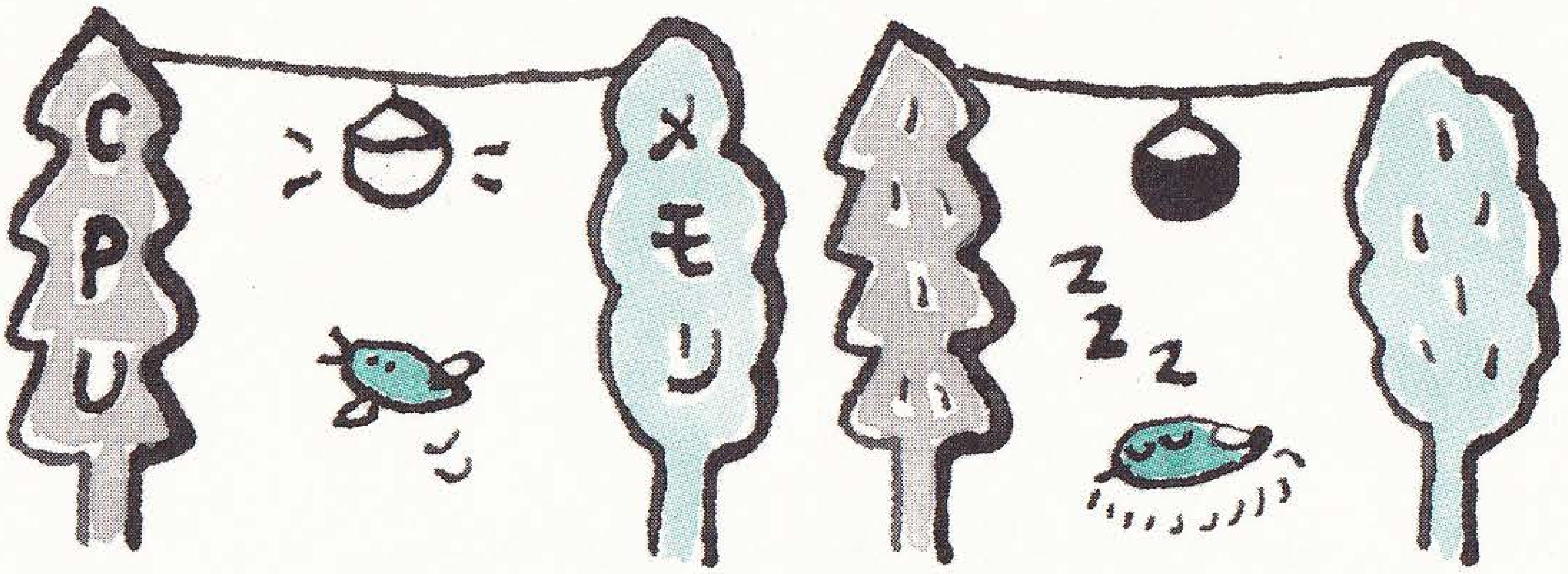
つまり電球の電線は、
電流が流れていれば（光っている）　　ON（1）
電流が流れていなければ（消えている）OFF（0）
と表すことができる。



電流が流れているならON（1）、流れていないならOFF（0）

このようにデータは、ON（1）とOFF（0）の電気信号に変換され、1本の電線でつながれることによって、CPUとメモリとの間でやりとりがされるんだ。

つまり、1本の電線でつながれた状態でデータが流れることで、コンピュータが電気信号を「**認識**」することができ、「**動く**」ということを可能にするんだ。



ONとOFFのデータはCPUとメモリ間で行われる

ポイント

- データは、ON（1）とOFF（0）の電気信号に変換される
- コンピュータは変換された電気信号を認識し、処理が行われる

「へえ、データは、ONとOFFの2種類を使って表現することができるのね。」

そうだ。それに、この**ONとOFF**をたくさん準備すれば、CPUが扱うどんな数でも表現できるんだ。

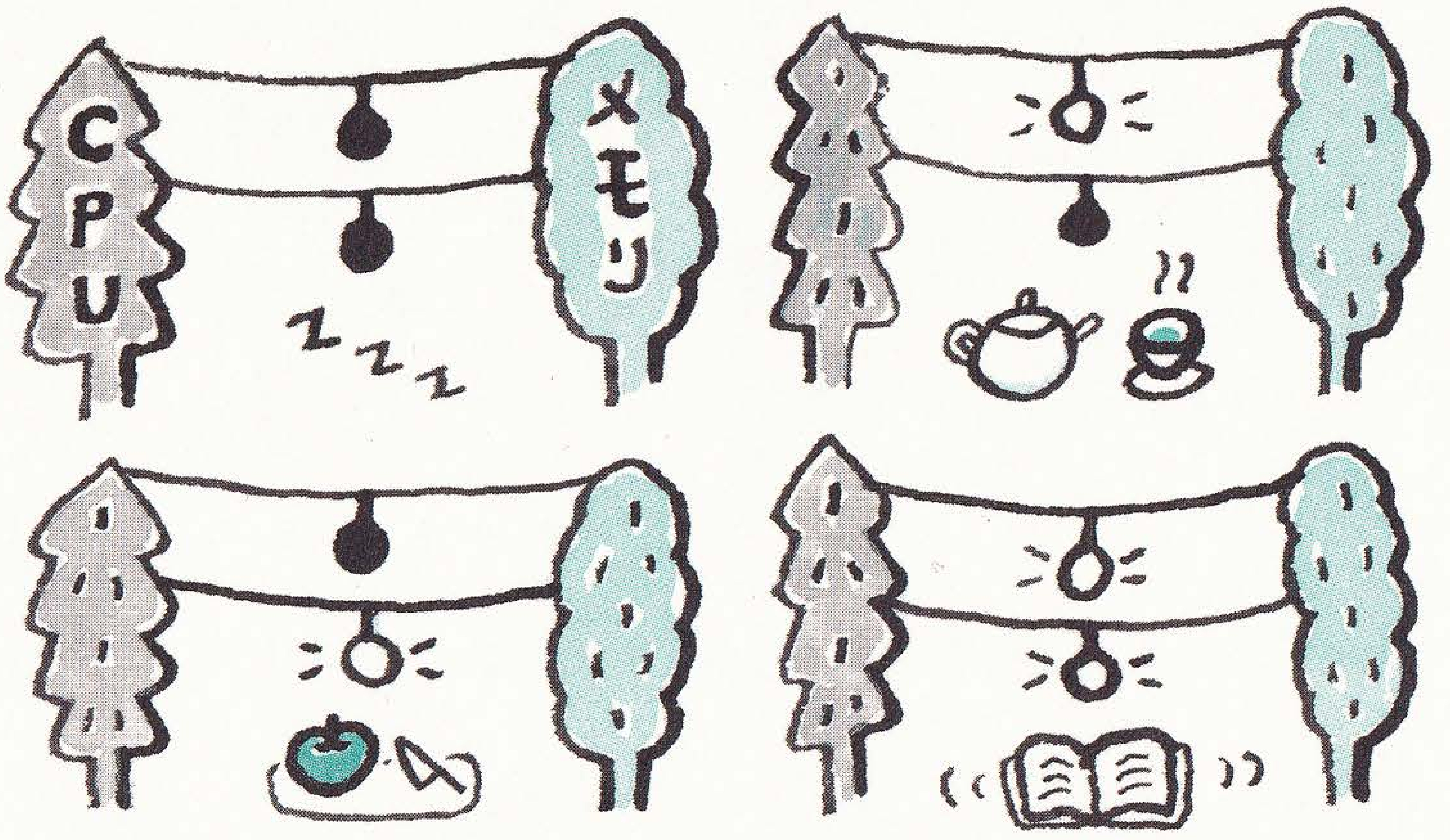


ONとOFFがあればどんな数でも表現できる

「どんな数でも表現できるって、どういうこと?」

1本の電線はCPUとメモリがつながっているね。
たとえば、この電線の数を1本から2本に増やすことで、「0」から「3」までの数が表現できるんだ。

（1本目の電線）	（2本目の電線）	（表す数）
OFF	OFF	0
OFF	ON	1
ON	OFF	2
ON	ON	3



1本目と2本目の電線で表す

つまり、
CPUとメモリがつながっている1本の電線を基準に考えて、
この**電線の数を増やすことによって、伝えるデータ量を増やすことができる**んだ。

基本的には、1本の電線の接続を単位として考えるけれど、
実際の仕事には電線をたくさん準備しておいて、CPUとメモリをつなげているんだ。
そうすることによって、それぞれの電球のONとOFFの信号を組み合わせて、効率よく仕事ができるんだ。

ポイント

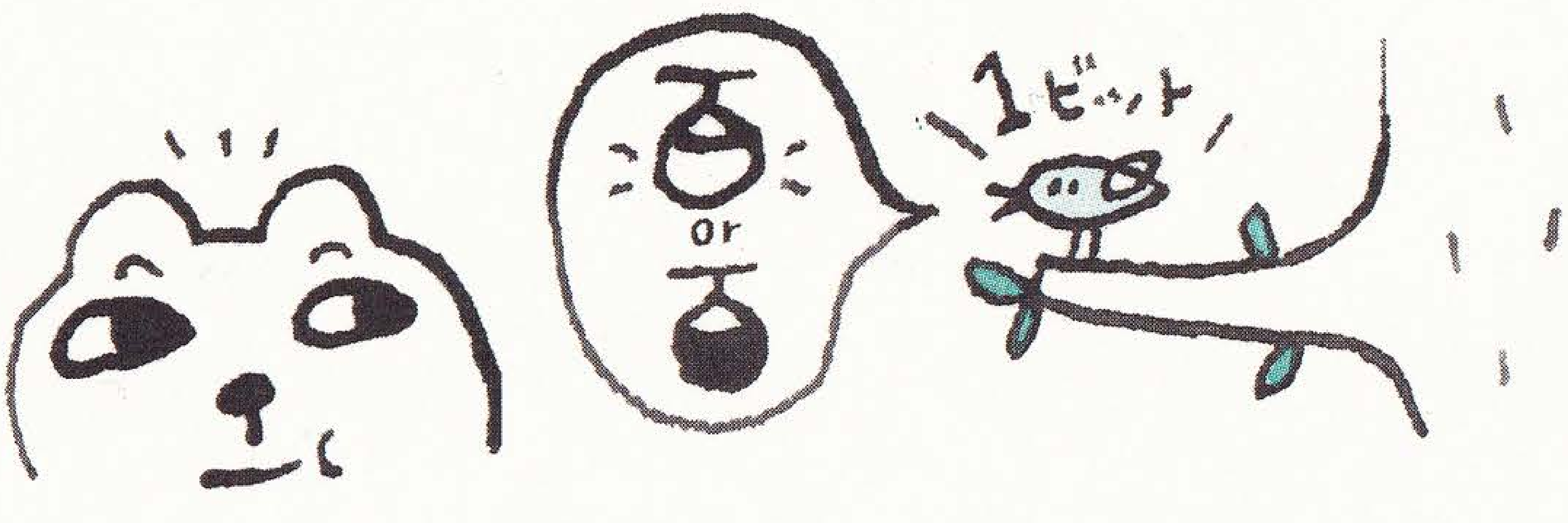
- データを伝える電線は、1本を単位の基準として「ON」と「OFF」を表現している
- コンピュータは「ON」と「OFF」を伝える1本の電線を基準とし、電線数を増やすことで、効率よく仕事をしている

ビットとバイト

ここでは、「ON」と「OFF」で表すことのできるデータの単位について考えていこう。

人間の命令はONとOFFの電気信号に変換されて、
コンピュータに情報を伝えているということだったね。
そして、伝える情報量を増やすには電線1本から2本、3本…というように電線の数を増やす（P48参照）ことで解決できた。

このON（1）とOFF（0）の2種類のデータを示すことのできる、
電線1本分のデータの単位を1ビットというんだ。



電線1本分のデータが1ビット

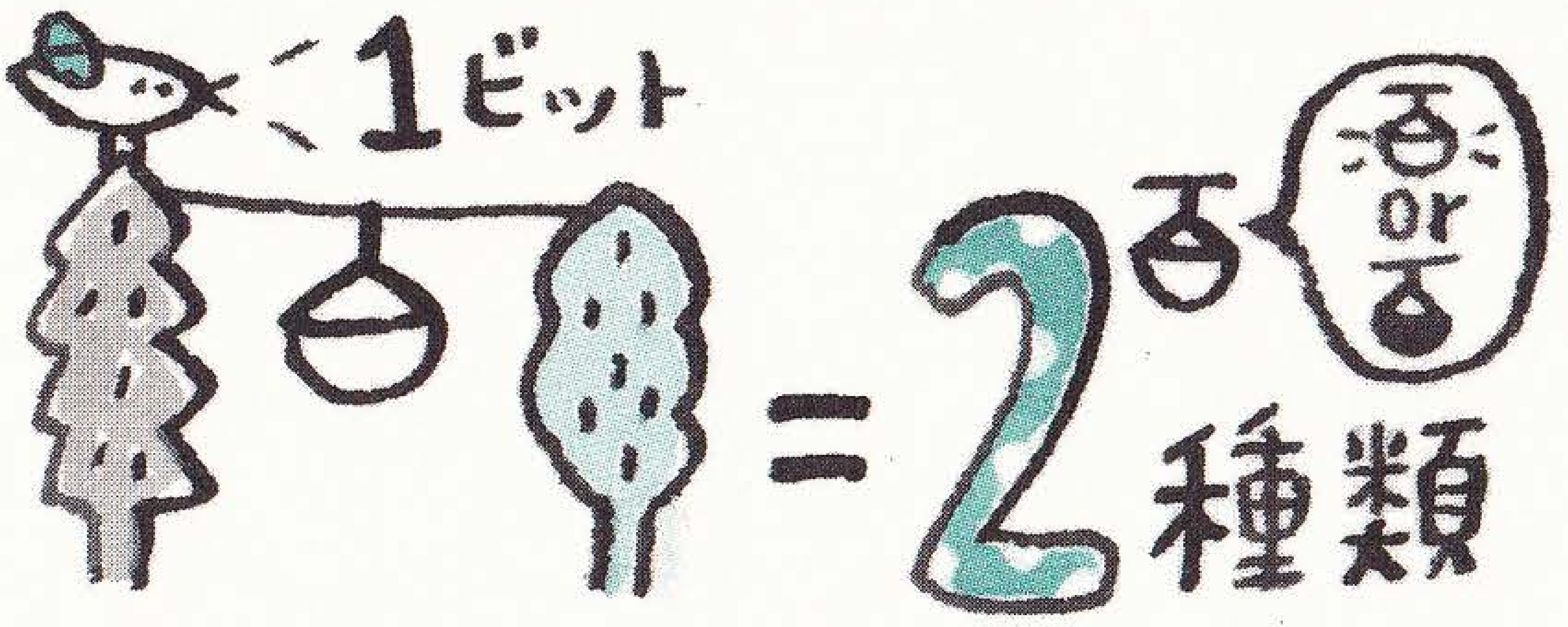
「**びっと**って何？」

ビットというのは、コンピュータが扱う、**情報量を示す「最小単位」**のことをいうんだ。

電球をつなぐ電線が1本の場合は、

電球1	表す数
OFF	0
ON	1

の（0,1）の2種類のデータを伝えることができる。
これが1ビットというわけだ。

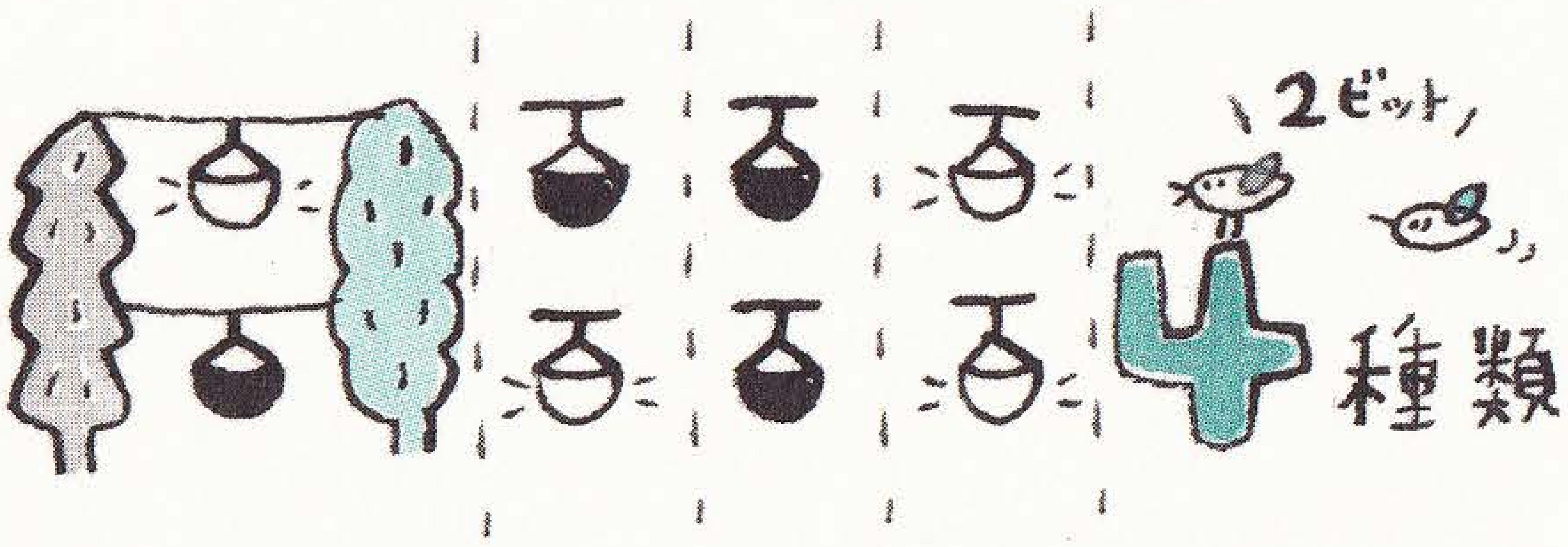


1ビットは2種類のデータを伝える

そして電線が2本の場合は、

電球1	電球2	表す数
OFF	OFF	（0,0）
OFF	ON	（0,1）
ON	OFF	（1,0）
ON	ON	（1,1）

の（0,0）（0,1）（1,0）（1,1）の4種類のデータを伝えることができる。
これが2ビットの情報量となる。

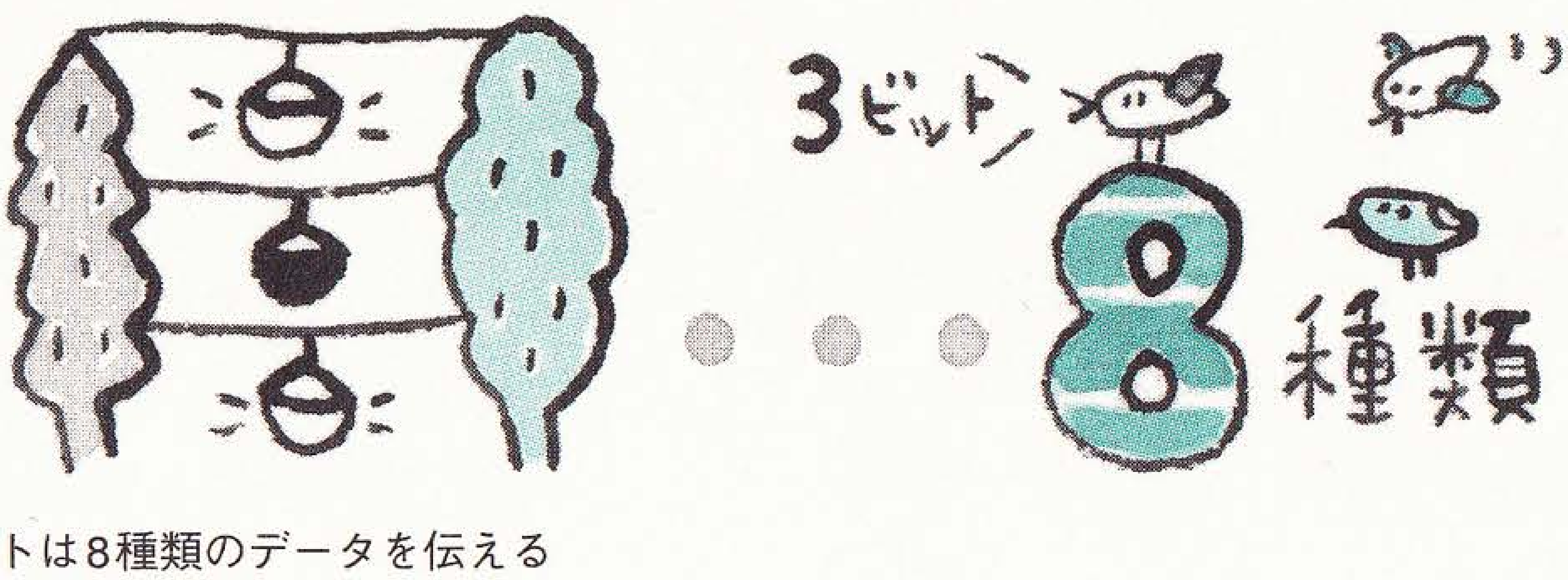


●2ビットは4種類のデータを伝える

電線を3本つなぐと、

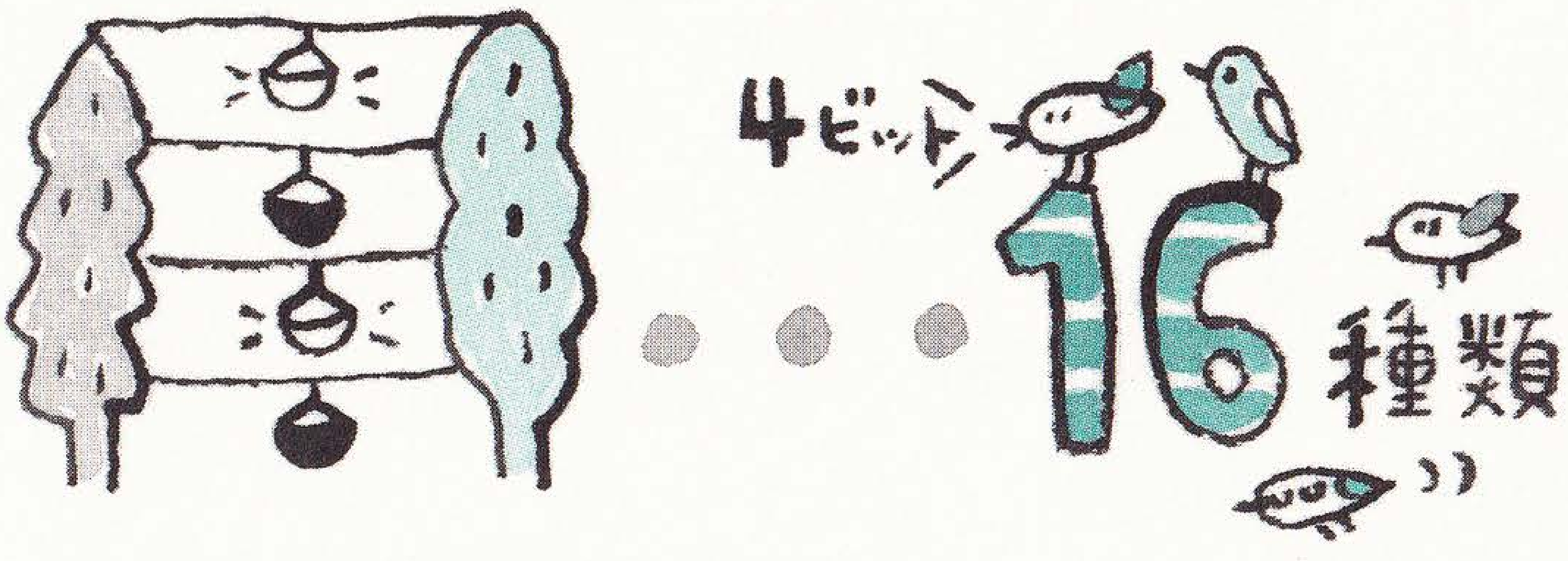
電球1	電球2	電球3	表す数
OFF	OFF	OFF	(0,0,0)
OFF	OFF	ON	(0,0,1)
OFF	ON	OFF	(0,1,0)
OFF	ON	ON	(0,1,1)
ON	OFF	OFF	(1,0,0)
ON	OFF	ON	(1,0,1)
ON	ON	OFF	(1,1,0)
ON	ON	ON	(1,1,1)

の8種類の情報量を表すことができる。
これは3ビットの情報量になるね。



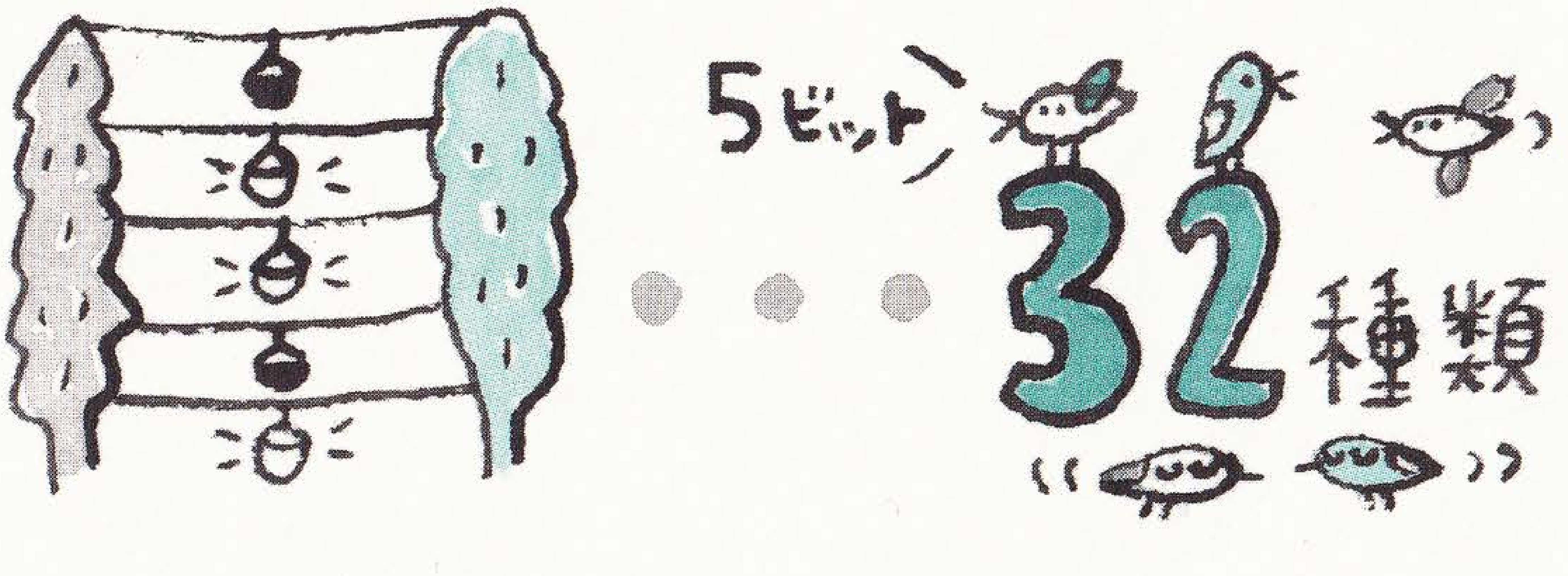
●3ビットは8種類のデータを伝える

同じように、電線を4本つなぐと、0、1、2、3、4、5、6、7、･･･、15で16種類の情報量を表すことができる。
これは、4ビットの情報量になる。



●4ビットは16種類のデータを伝える

また、電線を5本つなぐと、
0、1、2、3、4、5、6、7、･･･、30、31
で32種類表すことができる。
これは5ビットの情報量になるんだ。



●5ビットは32種類のデータを伝える

このように、
1ビットは、2種類の情報量を表し、
2ビットは、4種類の情報量を表している。
同じように、3ビットは8種類の情報量を表していることがわかる。

さて、このように見ていくと、ある一定の法則が考えられるんだ。
つまり基本的にnビットの情報量は「2のn乗通り（2n）」で示すことができる。
すると、4ビット、5ビットはどうなるかな?



「ええと…。4ビットは、2の4乗で16種類の情報量で
5ビットは、2の5乗で32種類の情報量かな…。」

その通り。
つまり、「**nビットの情報量は2のn乗通り**で表すことができる」といえる。

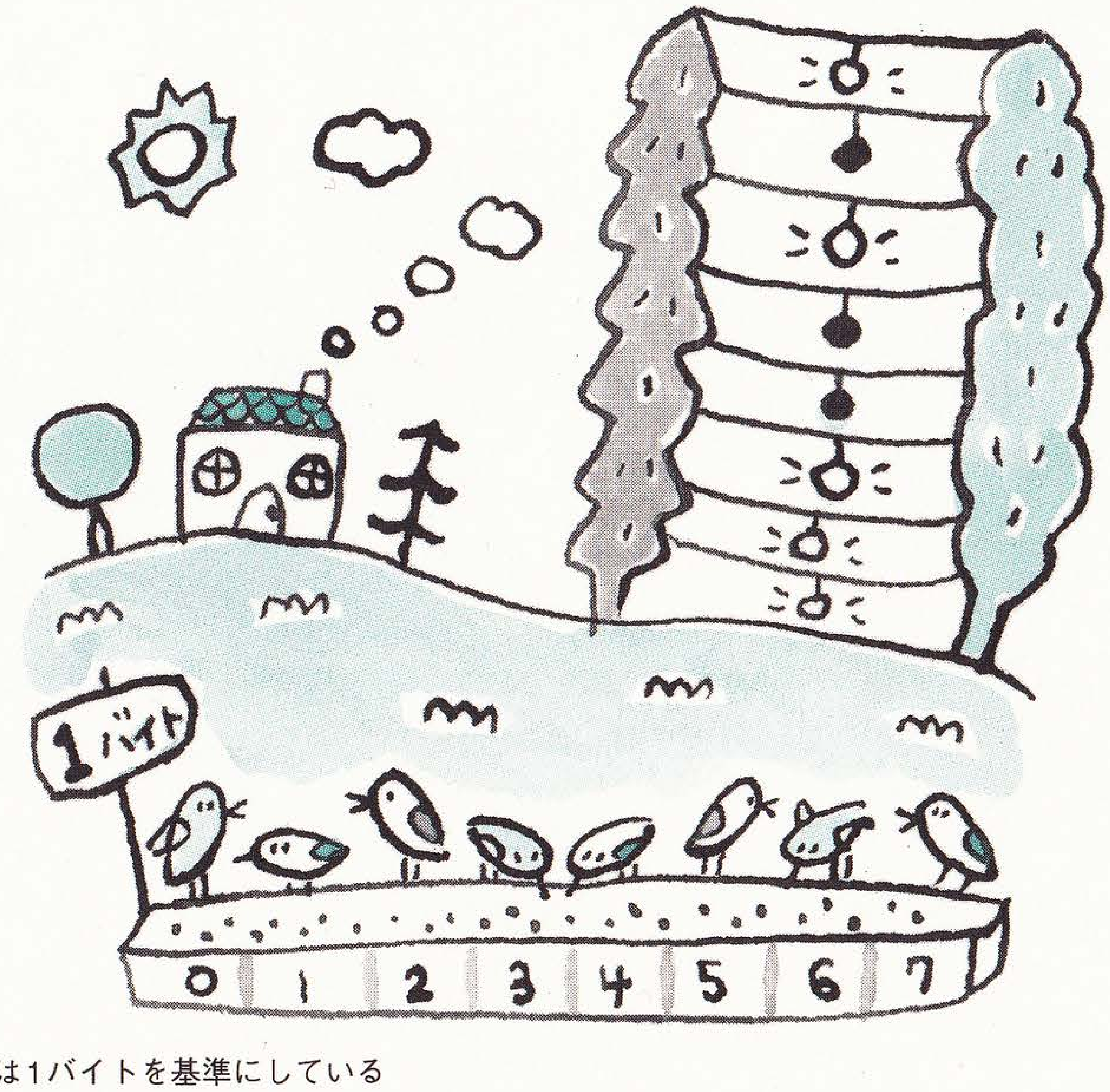
この基となっているのが2進数を使った考えなんだけれど、
2進数については、64ページで説明するね。

さて、ビットという単位については理解できたかな…?
それでは今度はデータの基準となる単位についてみていこう。

「データの基準となる単位って?」

それは、**バイト**と呼ばれるものだ。
バイトは、8ビットを基準にまとめたもので、
1バイト=8ビットとして考える。

「データは、1バイト(8ビット)を基準にしているんだね。」



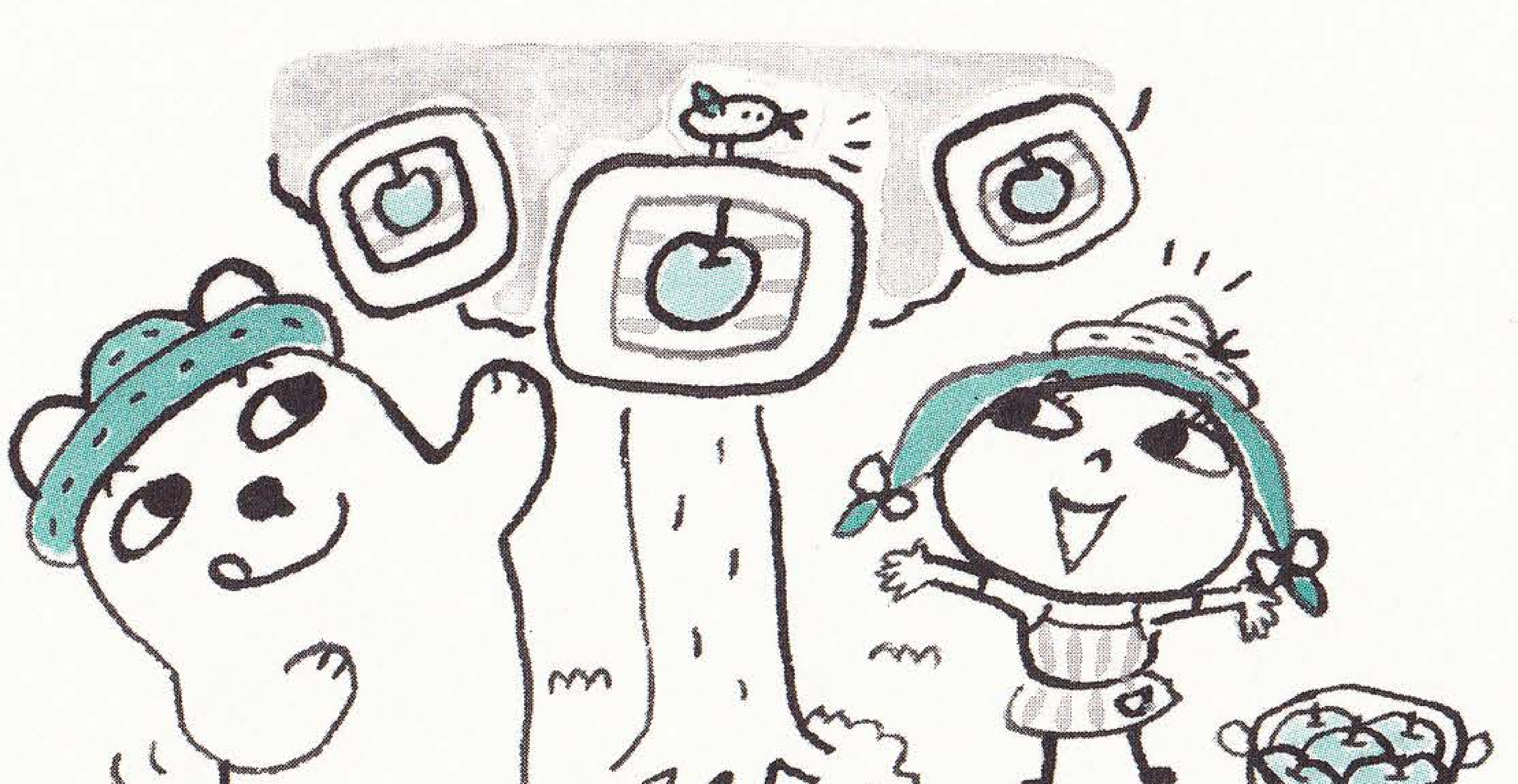
データは1バイトを基準にしている

8ビット以外にも
CPUの技術的な進歩によって、
16ビット、32ビット、64ビット単位で計算できるものなど、
一度に計算できる数がどんどん増えてきている。

そのため、大きくなったデータを単位として、
まとめた呼び方をすることがある。
たとえば、8ビット以上のデータの基本になる単位はバイトとして考えるのだけれど（8ビットは1バイトというように）、
2バイトをワード、
4バイトをロングワード
と考えると扱いやすい場合があるんだ。

ポイント

- コンピュータが扱う情報量の最小単位を1ビットという
- コンピュータは、1バイト=8ビットを基準として処理を行っている
- CPUの技術的な進歩によって、基準となる単位も増えてきている



2-3 進数とは何か

進数とは

コンピュータはデータの単位、1バイト（8ビット）を基準として、処理を行っていたんだね。
これらのデータの単位を考える基となる、**進数**について見ていくことにしよう。

「しんすう?　言葉は確か聞いたんだけど進数って何だろう?」

まずは、普通に数字というものを考えてみよう。

0から順番に数えていくと、9の次が10になる。
このままどんどん数えていくと、右側の数つまり、一桁目は、0から9までの数字を繰り返す。

繰り返すごとに左隣の数つまり二桁目に1が足されていく。
この関係はわかるかな。



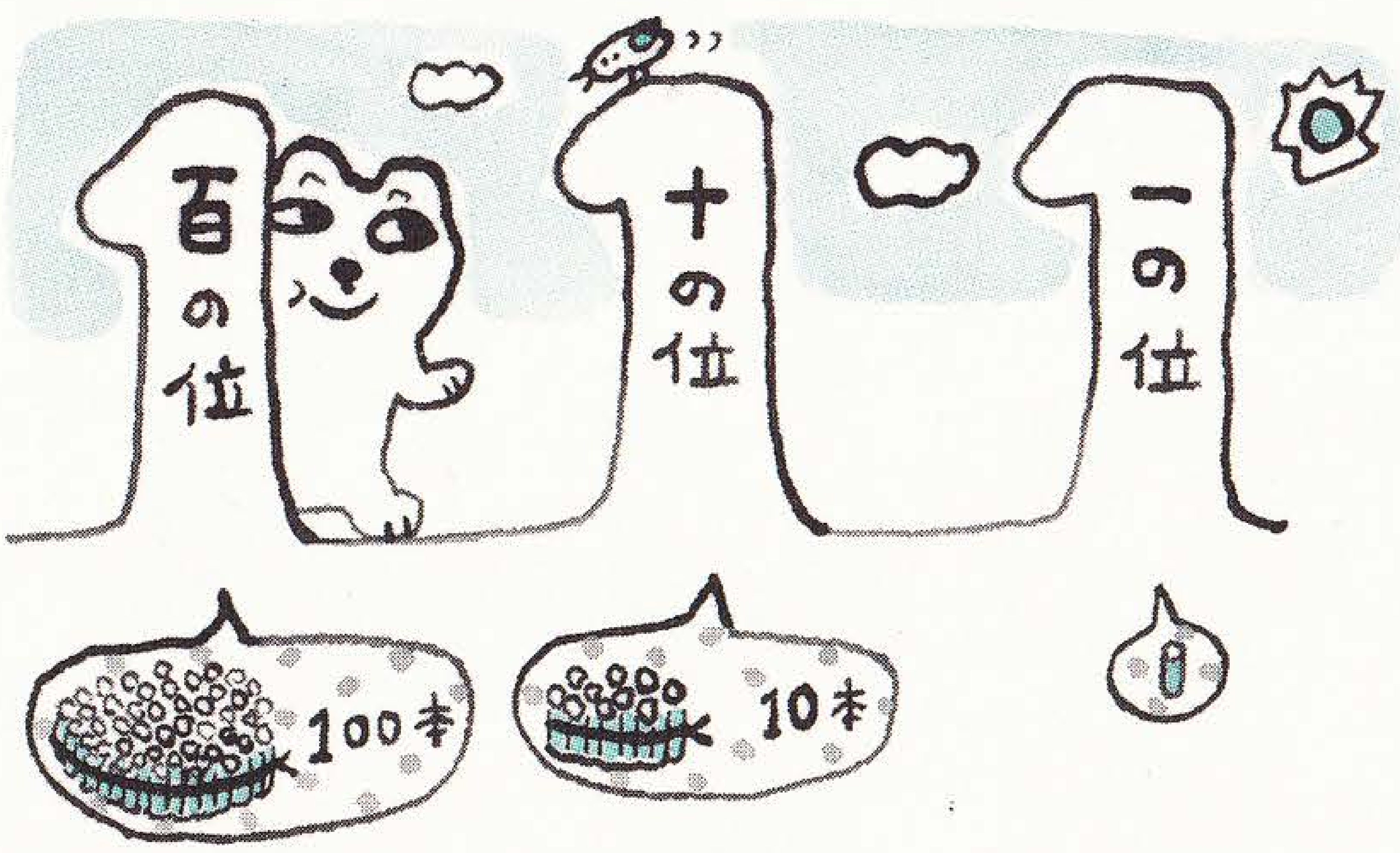
●繰り返すごとに左隣の数に1が足されていく

**「9の次は10でしょ、99の次は100だから、
なんとなくわかる。」**

この0から9までの10種類の数で表現する方法を**10進数**というんだ。

初めて数字を習った時のように数え棒で考えよう。

一の位は、棒そのものの数を示す。
十の位は、10本の棒の束数を示す。
百の位は、100本の棒の束数を示す。



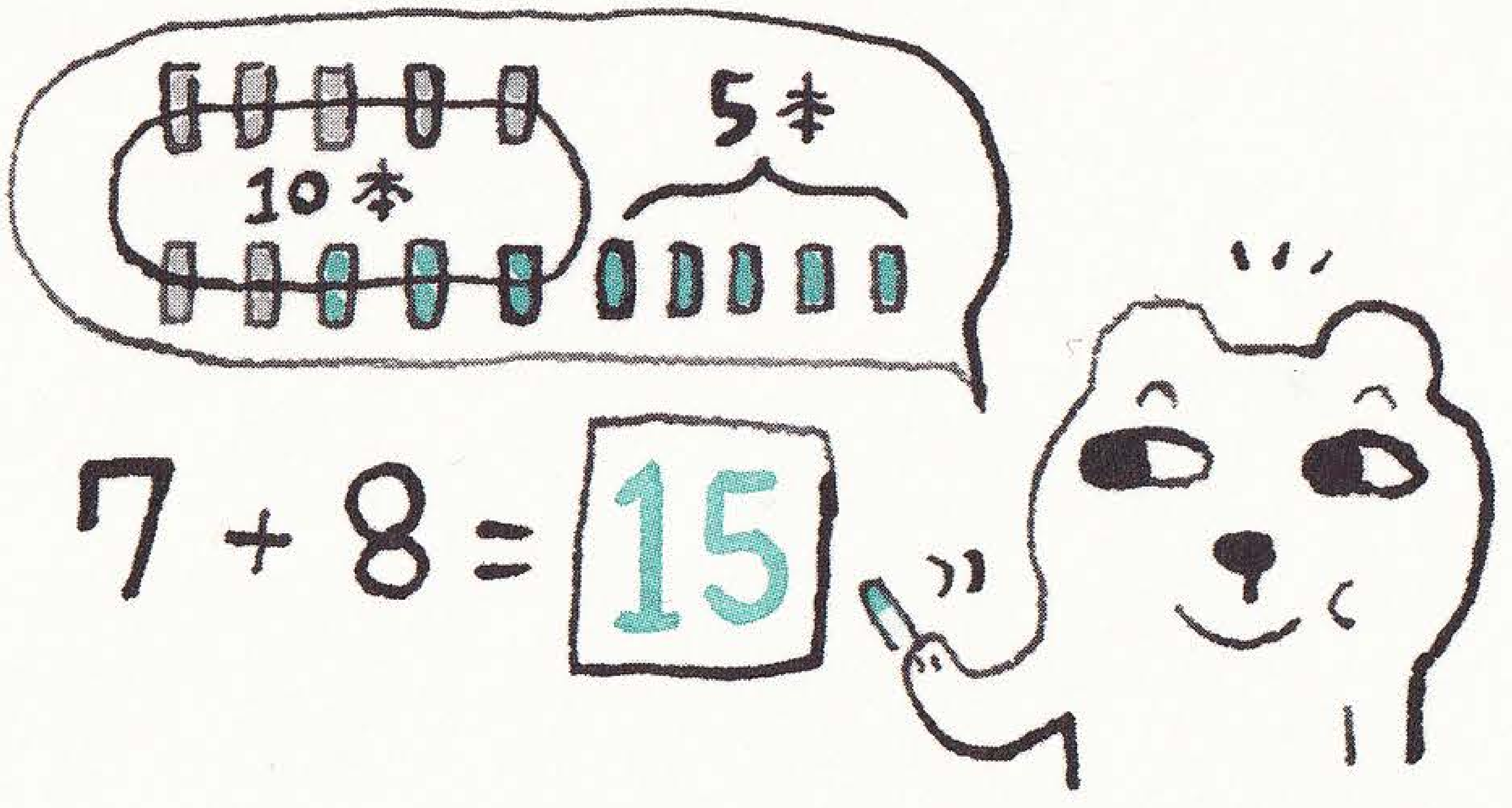
●棒を束ねた時の位の関係

というように、どこまでも数が進んでいく。これが10進数の考え方だ。
10進数の場合には、一の位でも十の位でも、百の位でもどんな位でも、
1つの位で示すことができる数字は最大9だね。
1つの位に10集まると、あっという間に束ねられて、
次の位の1として扱うんだ。

「輪ゴムを持って、誰かが構えている感じです。」

その束ねるということが重要なんだ。
たとえば、7足す8を計算するときに数え棒をイメージしてみよう。

基の数7に、8本の数え棒を足していくんだ。
すると、3本目を足したところで、10本の束ができあがる。
これで、あっという間に輪ゴムで束ねられて、残った本数は、5本。
10本の束1つとバラが5本で15本となる。



●10進数

「懐かしい感じがする。でも、それがどうして大事なの?」

この10進数の考え方は、私たちが計算する時に用いる。
ただし、コンピュータの世界では、今説明した10進数ではなくて、
2進数や**16進数**を使って処理をしているんだ。

10進数の例では、10本集まったらすぐに輪ゴムで束ねたけど、
2進数では、2本で束ねることになる（2進数のことはP64参照）。



2進数

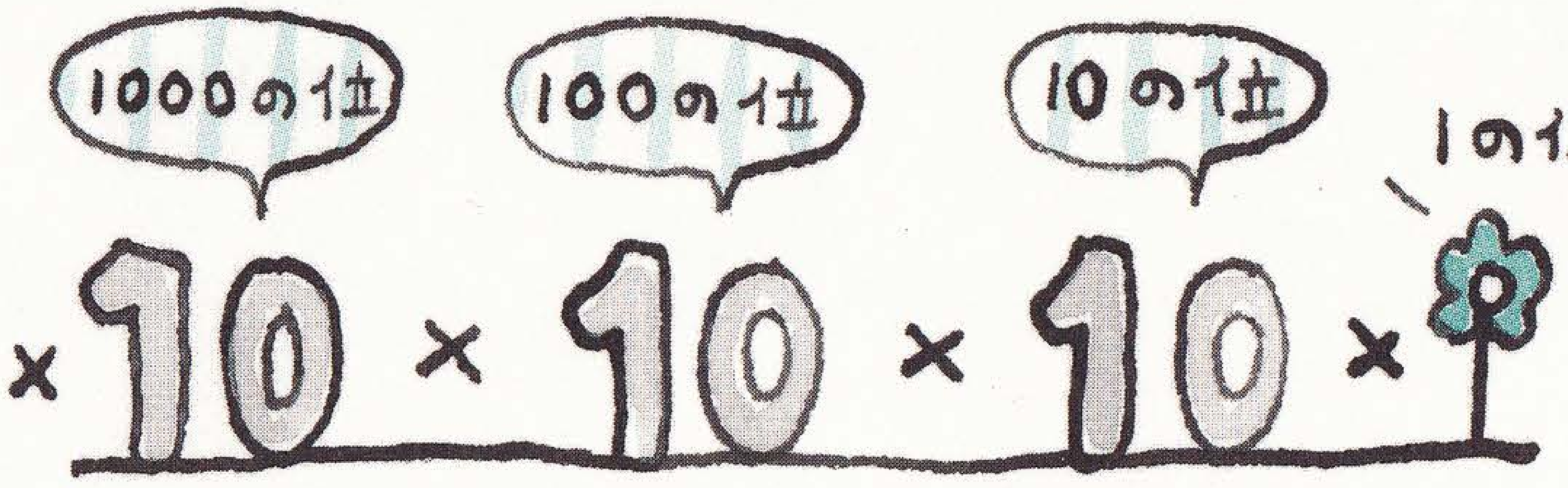
「ということは、16進数なら16本で束ねるのね（P67参照）。」



16進数

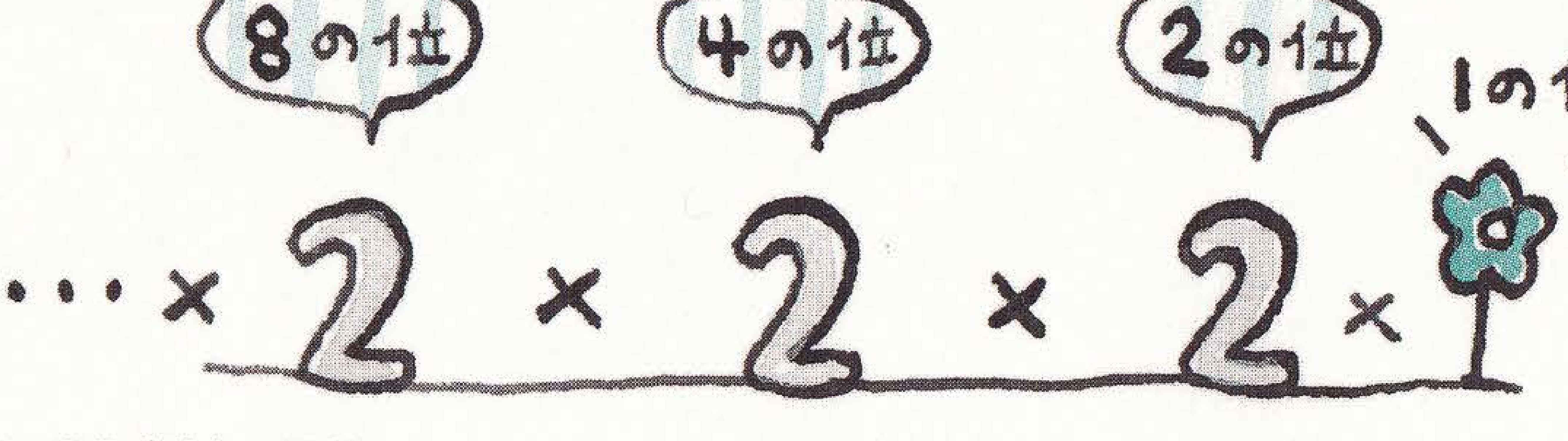
そういうことだ。
束ねた結果は、10進数の時と同じように、次の位に数えられる。
ここで、束ねる基本となる数を**基数**というんだ。覚えておこう。

10進数なら10が基数で、右の位から順に、
1の位、10の位、100の位、1000の位となるわけだ。
左隣の位つまり二桁目は、右隣の位の10倍になっている。



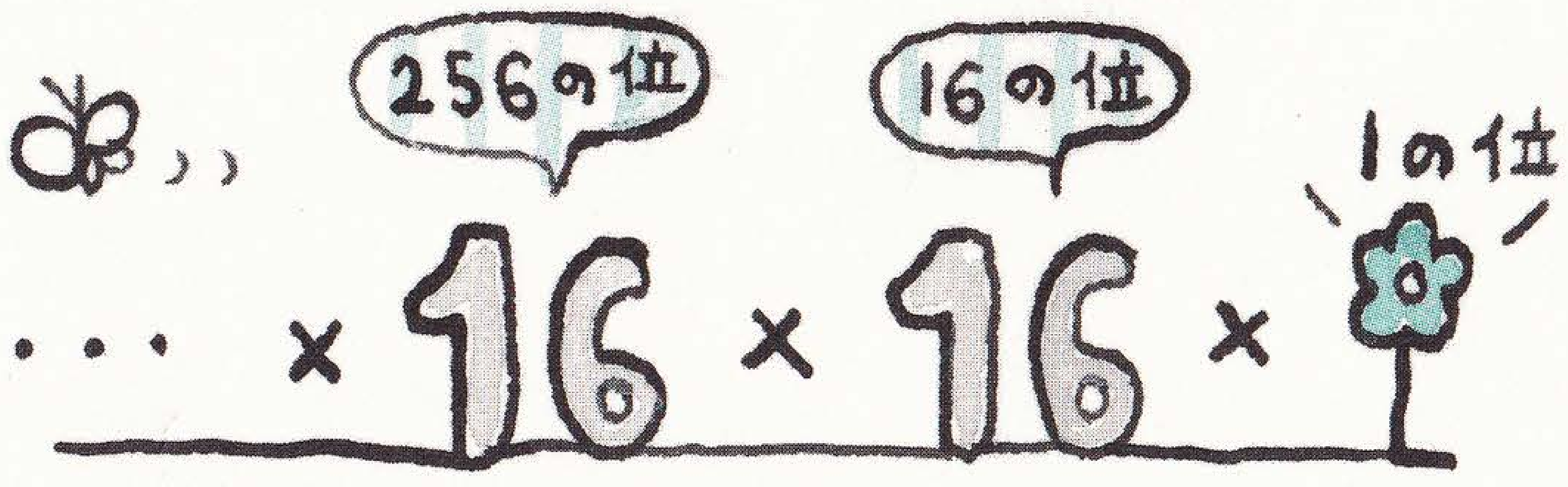
10進数「位」の関係

2進数なら2が基数で、左隣の位つまり二桁目は、右隣の位の2倍になっている。
右の位から順に、1の位、2の位、4の位、8の位となるわけだ。



2進数「位」の関係

「16進数なら、16倍だから、1の位、16の位、256の位になるわけですね。」



16進数「位」の関係

くま君、わかってきたね。
この3種類の進数の位を乗数で書くと、もっと簡単に表現できるんだ。

乗数

進数	1桁目	2桁目	3桁目	N桁目
10進数	10^0	10^1	10^2	10^{n-1}
2進数	2^0	2^1	2^2	2^{n-1}
16進数	16^0	16^1	16^2	16^{n-1}
M進数	M^0	M^1	M^2	M^{n-1}

ポイント

・進数とは、基準にする数まで数えたら、桁上がりをする考え方である
・進数では、基数をMとした場合、1桁目はM^0を示し、N桁目はM^{n-1}を示す

2進数

進数の考え方は、わかってきたかな?
ここではコンピュータが扱う2進数について考えていくことにしよう。

データは、
ONとOFFの電気信号に変換されてコンピュータに認識されている
ことがわかったね(P46参照)。

このON(1)とOFF(0)の2種類の信号で表すことのできるデータ
を**2進数**という。

つまり、ONとOFFの電気信号の処理は、
2進数で表現できるということなんだ。


「0と1で表現できるのが2進数ということね。」

そして、この2進数で表現された数値データによってコンピュータは
情報を扱うことができるんだ。
このときコンピュータは、CPUとメモリとの間を
4本の電線でつないだときに示すことのできる16種類の数(4ビット)
を基準にデータをやりとりしている(P67参照)。

4本の電線のONとOFFの組み合わせで示される下の表をみてみよう。

電線数と進数の関係

1本目	2本目	3本目	4本目	2進数	10進数
OFF	OFF	OFF	OFF	0000	0
OFF	OFF	OFF	ON	0001	1
OFF	OFF	ON	OFF	0010	2
OFF	OFF	ON	ON	0011	3
OFF	ON	OFF	OFF	0100	4
OFF	ON	OFF	ON	0101	5
OFF	ON	ON	OFF	0110	6
OFF	ON	ON	ON	0111	7
ON	OFF	OFF	OFF	1000	8
ON	OFF	OFF	ON	1001	9
ON	OFF	ON	OFF	1010	10
ON	OFF	ON	ON	1011	11
ON	ON	OFF	OFF	1100	12
ON	ON	OFF	ON	1101	13
ON	ON	ON	OFF	1110	14
ON	ON	ON	ON	1111	15

この表では、4本の電線に流れる電気のONとOFFの組み合わせと、この組み合わせを2進数で表したもの、10進数で表したものの、それぞれの対応を示している。

たとえば、4本の信号がそれぞれ、「ON-OFF-ON-ON」の場合をみてみると、2進数では、「1011」10進数なら「11」となっているね。

この表のように、電気信号は2進数や10進数などの進数で表現できる。
私たちは、普段10進数を用いるけれど、コンピュータが実際に情報として扱う時には、2進数で理解をしている。

つまりコンピュータは、どんな情報でも、電気信号のON（1）とOFF（0）で表現できる2進数を使ってうまく情報を伝えている、ということがわかるね。

ポイント

- 2進数とは0と1だけで表現する数のことである
- コンピュータで扱うデータは2進法のONとOFFに対応し、電気的に処理を行っている

16進数

ここではコンピュータが理解するだけでなく、
人間がプログラミングをする際に用いる、**16進数**について考えていこう。

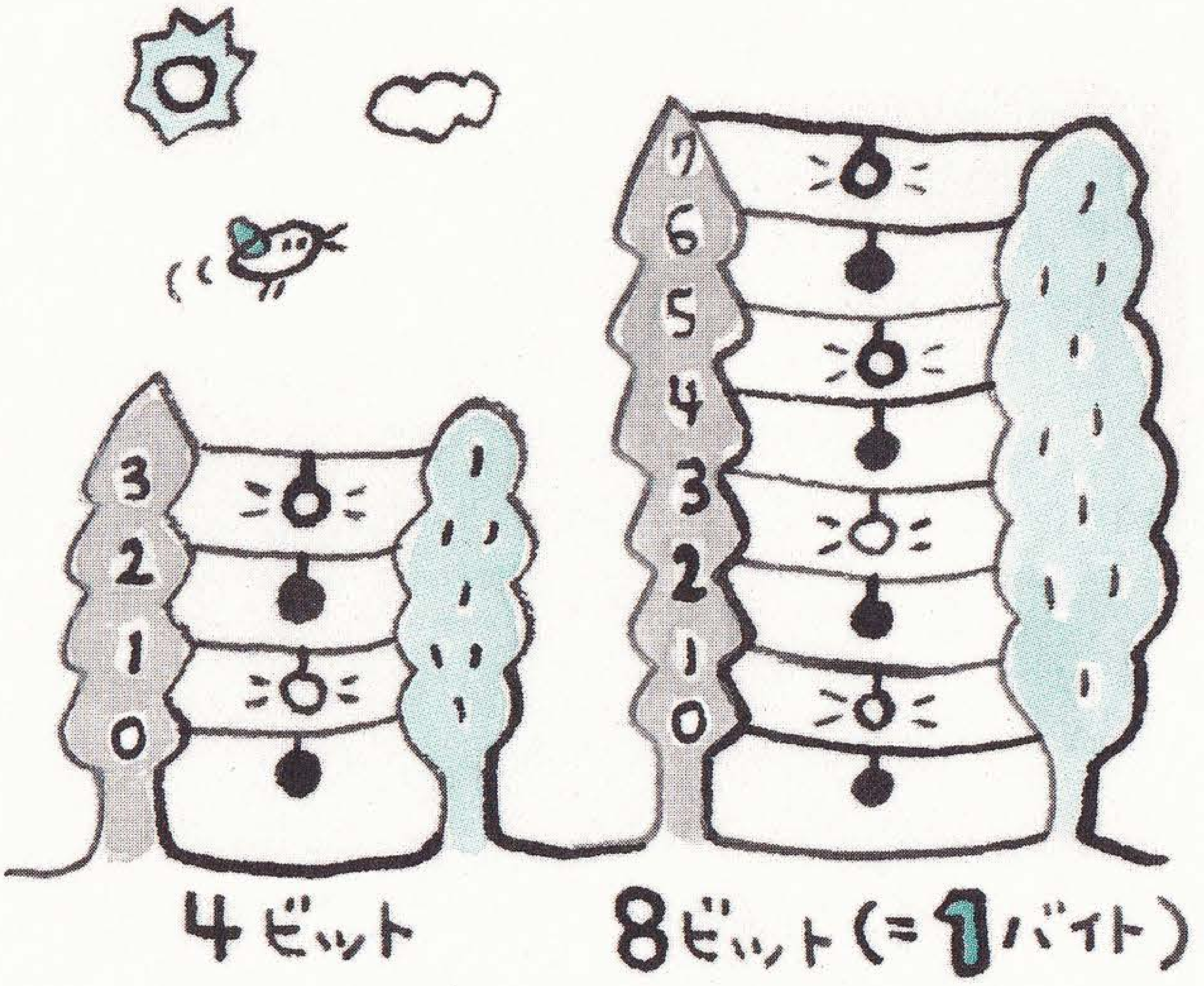
16進数は4本の電線のONとOFFの組み合わせを考えた数で、
0、1、2、3、4、5、6、7、…15の
16種類の情報量で表すことのできるデータのことをいう。
これは**4ビット**を表すね（P53参照）。

この16種類の情報量（4ビット）が16進数を考える上で基準となっていて（**基数**）、プログラミングを行う時にアドレスやデータを表わすのに使われる。16進数は、2進数よりもわかりやすく、10進数より大きな数を扱えるので、プログラミングするには便利なんだよ。
ただ、データは、1バイト（8ビット）を処理単位の基準としている（P55参照）ために、実際のプログラミングでは、この4ビットを2つ組み合わせて、8ビット（1バイト）として扱っているんだ。


「どういうこと?」

つまり、同時に認識できるデータ量を大きくした方が
効率よく作業ができるので、
16進数の1桁の4ビットではなく、**16進数の2桁を8ビット**として、
ひとつの基準としたんだ。

4ビットの時は、電線が4本あったけれど、8ビット（1バイト）の時には、この4本の電線の束をもうひとつ追加して、2倍の8本になるということをイメージしてくれたらいい。



●4ビットは電線が4本、8ビットは電線が8本

このように考えて、
プログラミングでは16進数を基準に考えているんだ。

さて、16進数の表現方法をみていこう。
16進数で表現できる数は、0～15までの16種類で（P67参照）、
10～15までの数字は、10進数だと2桁になる。
しかし16進数で考えた時には、2桁だと扱いづらい。

具体的には、9より大きい数字を表現する場合に問題が起きてくる。
9の次は10になる。しかし進数としては桁上げをしたくないので、
16進数では、アルファベットを数字として使うことにしたんだ。

つまり、10進数の10を16進数ではA、11をB、12をC、13をD、14をE、15をF…と表現するようにしたんだ。

16進数

10進数	16進数
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
:	:

こうすると、16進数の数は、
0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、Fとなる。
F（=15）の次は、進数の基数16だね。

この時の16は基数なので、
16の束で桁が上がって10（いちぜろ）になるんだ。
この場合の10は、10進数と区別するために、「じゅう」と読まずに「いちぜろ」と読むことに気をつけよう。



●16進数の基準


「16進数は、16が基準となって桁上がりされているのね。」

そうだね。16進数は16本で束ねられるというわけだね（P61参照）。

ポイント

- 16進数は、プログラミングを行う時に用いられる進数である
- 16進数では、16を基数とし、10をA、11をB、12をC、13をD、14をE、15をFとして表現している

最後に2進数、10進数、16進数の対応をまとめておこう。

進数対応表

2進数	10進数	16進数
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10

進数の桁上げに注目すると、それぞれの進数の関係がわかるよね。
2進数の桁上がりは、1の次で10だよね。
これは、10進数なら2、16進数でも2だ。
10進数の桁上がりは、9の次が10だよね。
これは、2進数なら1010、16進数ならAになる。
16進数の桁上がりは、Fの次が10（いちぜろ）だよね。

これは、2進数なら10000、10進数なら16だ。
このようにコンピュータは、2進数で扱われているため、表のように、ある進数を2進数に変換してコンピュータが処理しやすいようにしているんだ。

また、すべての進数は互いに対応関係を持っている。わかりにくい時には扱いやすい10進数に変換することで、その関係を理解することができるんだ。

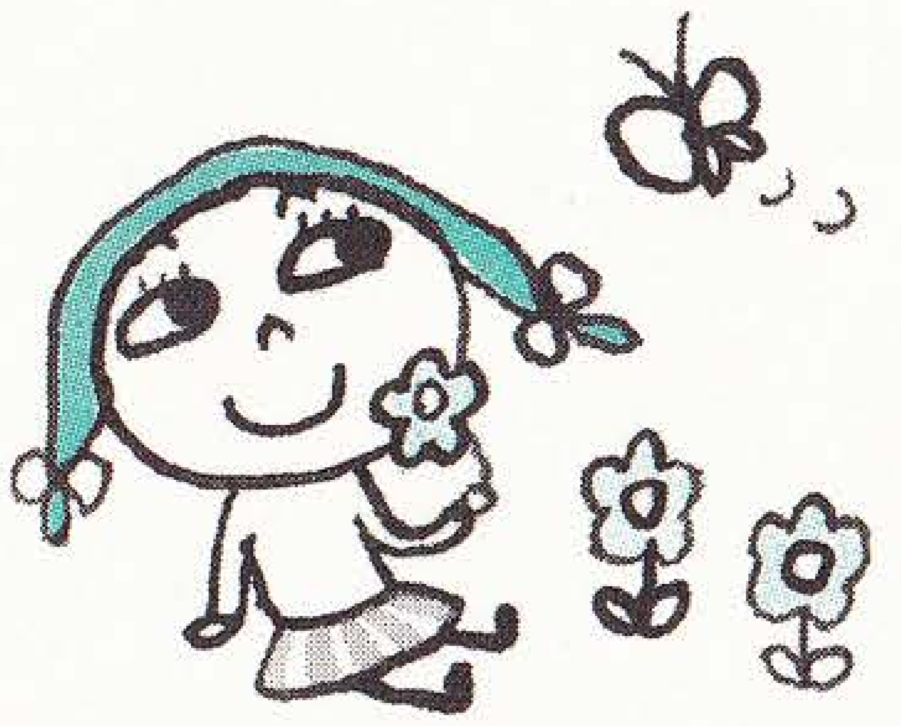
ポイント

・どんな進数でも、他の進数に換算できる
・進数を用いることで、コンピュータの処理やプログラミングは行われている
・他の進数に換算する場合には、10進数を基準に換算するとわかりやすい

まとめ

- コンピュータは、入力装置、演算装置、記憶装置、出力装置の4つの装置から構成されている
- コンピュータに与える命令は、すべてデータ化されて処理が行われている
- データは、すべてON（1）とOFF（0）の電気信号に変換されている
- コンピュータが扱う情報量の最小単位を1ビットという
- コンピュータは、1バイト（8ビット）を基準としている
- コンピュータは、2進数や16進数を用いて、処理やプログラミングを行っている

第3章
プログラミングのしくみ



絵美が今日の夕食のレシピは何がいいか、考えている。

絵美「今日は、どうしようかな…。カレーは昨日作っちゃったしなあ。今日は、シチューにしてみようかしら。」
くま「あまり、昨日と変わらないですね。」
絵美「贅沢なことをいわないでよ。私だって、色々忙しいんだから…。それに今日お父さんは、帰りが遅くなるっていっていたし、私とくま君の分で十分だから、昨日の残りの材料で使い回しができるでしょう。」
くま「色々と考えているんですね…。
それより、お腹空いたなあ。早く食べたい…。」

どこまでもマイペースでわがままな、くまであった。

絵美「くま君も夕食の準備を手伝ってちょうだい。」
絵美「私とくま君の仕事を分担して、誰がどの仕事をするのか決めよう。そうすれば、効率よく夕食の準備ができるよ。」

着々と夕食の準備が進み、おいしく夕食をいただいた、絵美とくま。

午前0時。ピンポーンとベルが鳴る。
どうやら、先生のお帰りだ。

先生「絵美、今日の夕食は何だった？　うーん。いいにおいがする。シチューだね。」
絵美「うん、そうなの。でも、くま君に文句をいわれたの。」

夕食ができるまでの一部始終を話した絵美。

先生「絵美は、えらかったね。まさしく、プログラミングをしたんだよ。」

絵美「えっ。そんなのしていないよ。私は、お料理をしただけなの。」

くま「お料理がプログラミング???　頭が混乱してきた…。」



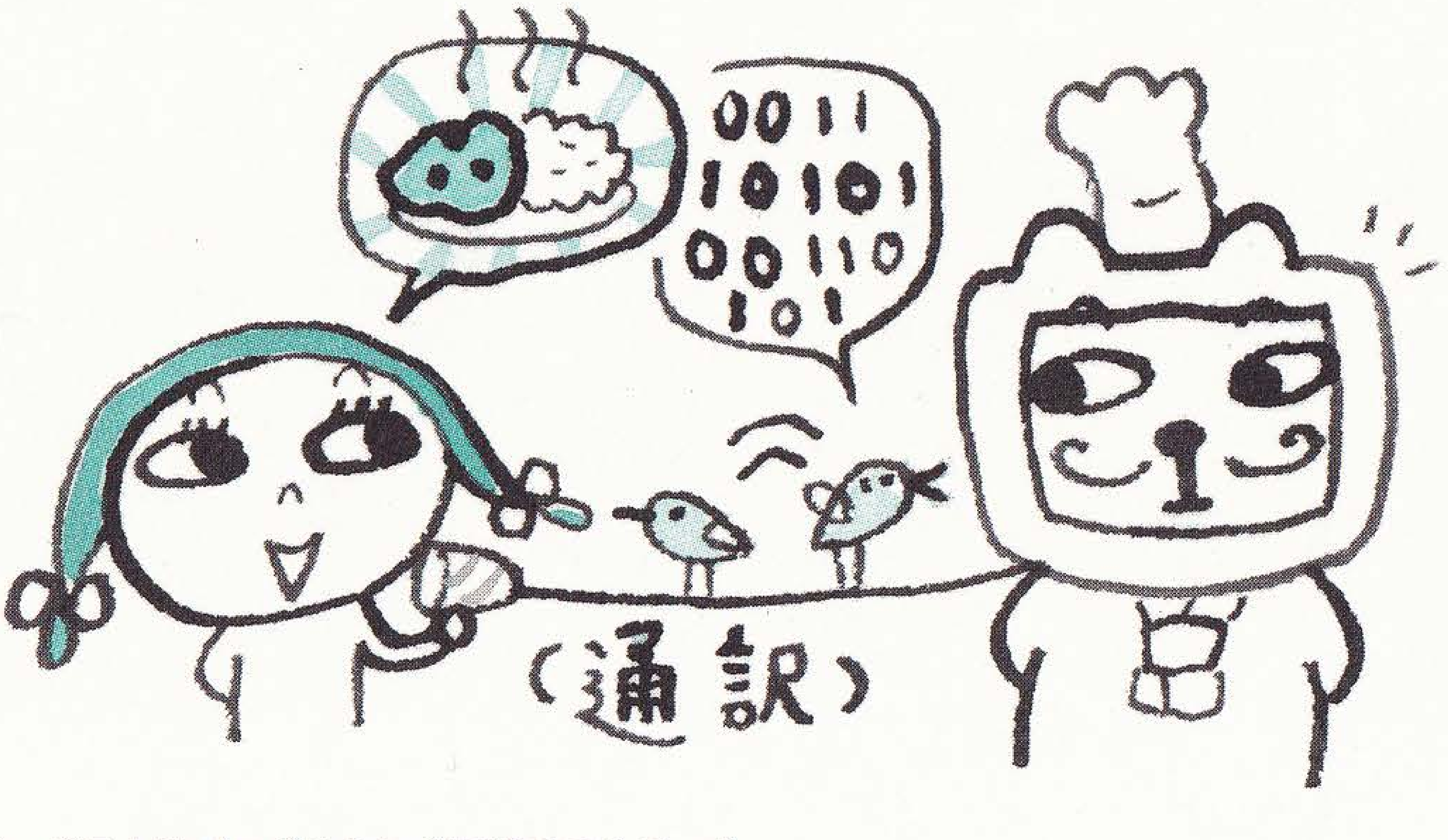
3-1 プログラミング言語の構成

プログラミング言語とは

お料理がプログラミングだなんて、信じられない、絵美とくま。

実は、料理することは、プログラムを作ることと同じなんだ。
まず、プログラミングするときには、レシピと同じように指示書が必要であったり、どういう目的で何を作るのかを考える必要がある（P84参照）。
まずは、プログラミングをする上で必要なコンピュータ言語についてみていこう。

私たち人間はコンピュータに指示を与えるとき、
プログラム（数値化されたデータ）を使って指示を出していたよね。
この数値化されたデータは**プログラミング言語**でできているんだ。



プログラムはプログラミング言語でできている


「プログラミング言語って？」

プログラミング言語は、
人間の命令が
コンピュータにわかるように、
あるいは、
コンピュータにとって理解できる言葉が
人間にもわかるように、
通訳の役割を果たしているんだ。
つまり、42ページで「コンピュータが情報を理解するには、
データ（数字）を**媒介**として、意思を伝え合う」としていたけれど、
この仲介となる言語がまさしく**プログラミング言語**と呼ばれるもの
なんだ。


「プログラミング言語は、人間とコンピュータとの間を取り持って、話を通じ合わせているんだね。」

そうだね。人間同士は、日本語、中国語、英語などが必要だ。


「ところで、プログラミング言語って一体どういうものがあるの？」

プログラミング言語には、大きく分けて2種類の言語がある。
1つは、**低級言語**と呼ばれるものでコンピュータが直接理解できる言語だ。

低級言語は、
単語の数や文法などが少なく、コンピュータに近い言語のため、
私たちが扱うには必ずしも効率的ではないんだ。
たとえば、機械語やアセンブリ言語がこの低級言語に当てはまる。

これに対し、**高級言語**と呼ばれるものがある。
これは、人間の言葉に近い言語といえる。
人間が理解できる言語のため、あまりコンピュータを意識しないで、
プログラミングしていくことのできる言語なんだ。
つまり、コンピュータにプログラムが
何を命令しているかを理解しながら、
プログラムを作成することができる言語なんだ。
たとえば、C言語やJava言語などがこの高級言語に当てはまる。

このようにプログラミング言語には、低級言語と高級言語があるわけだ。

コンピュータに指示することが少ない場合には、
低級言語でも十分なんだけど、
たくさんの仕事を指示しないといけない場合は、
プログラムが多くなってしまうため、
低級言語は、私たちにとって理解しがたいものになってしまうんだ。

そのため、一般的にプログラムを作成するときには、
高級言語がよく使われるんだ。
それは、低級言語が「**コンピュータを直接動かす**」という考え方で作られているのに対して、高級言語は「**人間がやりたいことをコンピュータに伝える**」という考え方で作られているためなんだ。



プログラム作成に使われるのは高級言語

ポイント

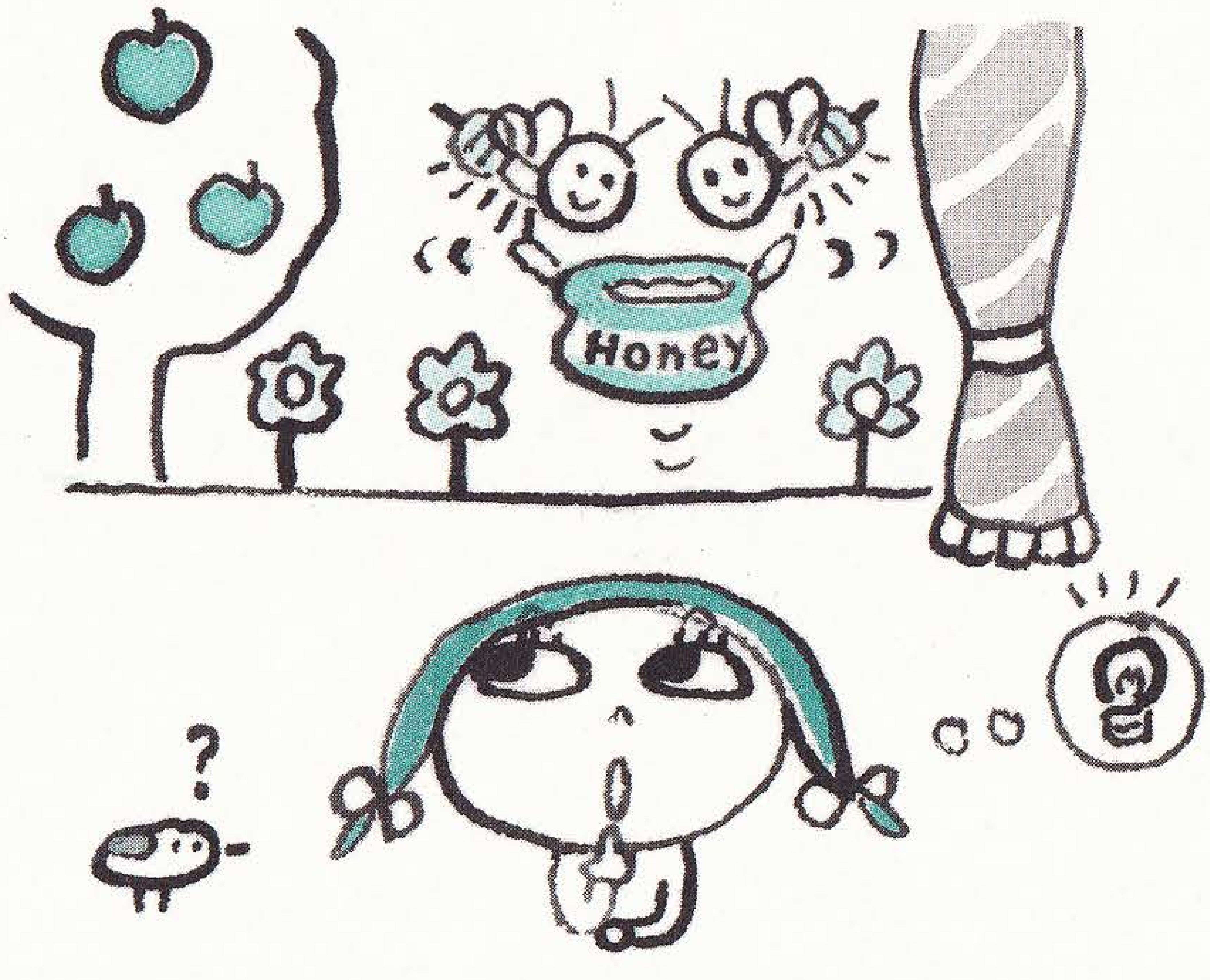
- プログラミング言語には、高級言語と低級言語がある
- 一般的にプログラムの作成に用いるのは、C言語やJavaなどの高級言語である

3-2 プログラミングの流れ

プログラミングの作業とは

プログラミング言語のことがわかったら、
早速、プログラムを作ってみたい。
でもその前に、
プログラミング全体の流れをみてみよう。

プログラムを作る場合、
まず、コンピュータにして欲しい仕事を考えて、
実際に**どういうプログラムを作成していきたいか**を考えるんだ。
この作業を**分析**という。



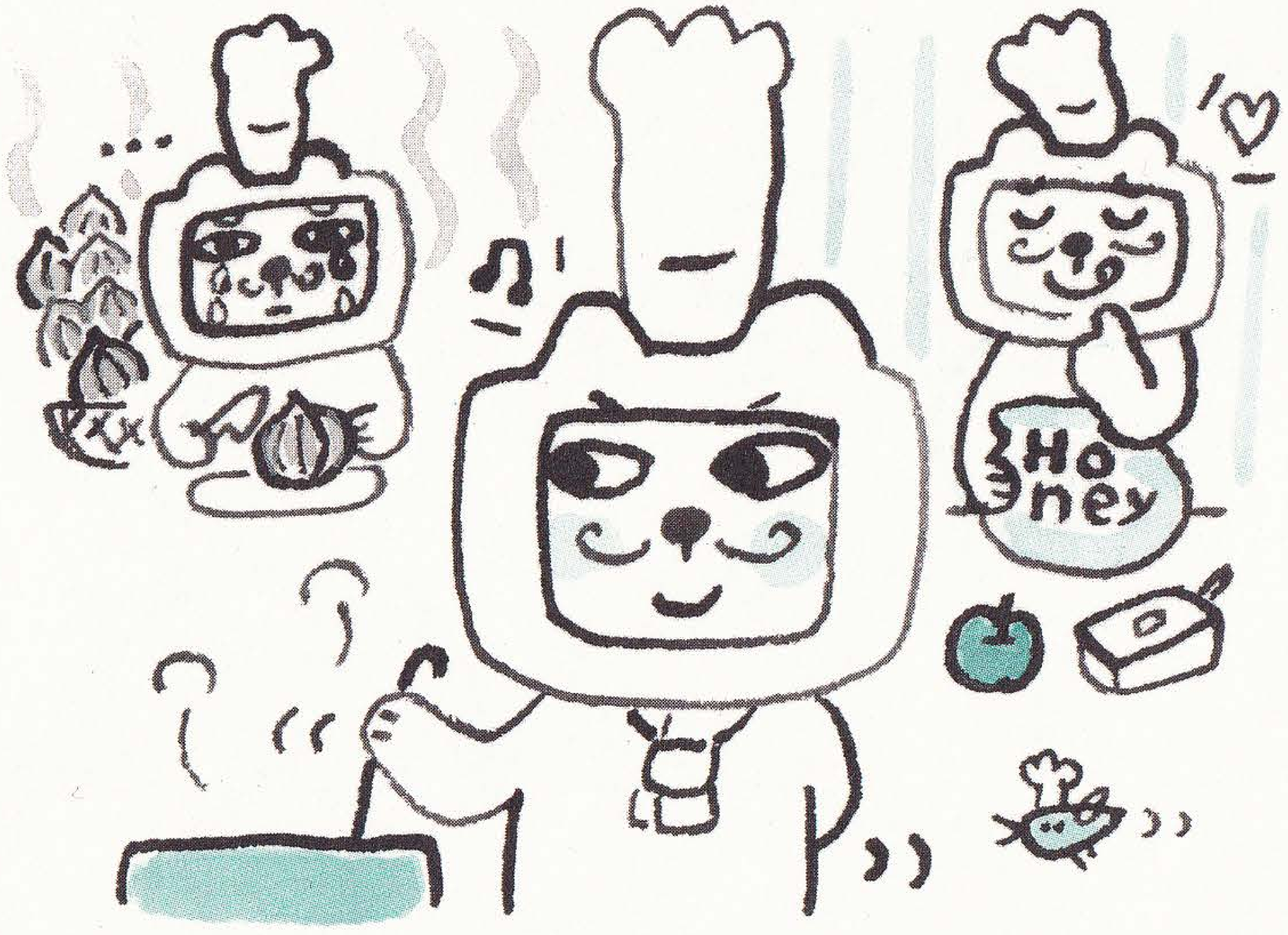
分析

この分析が確定した後に、
次に**プログラムの作り方**を考えるんだ。
この作業を**設計**という。



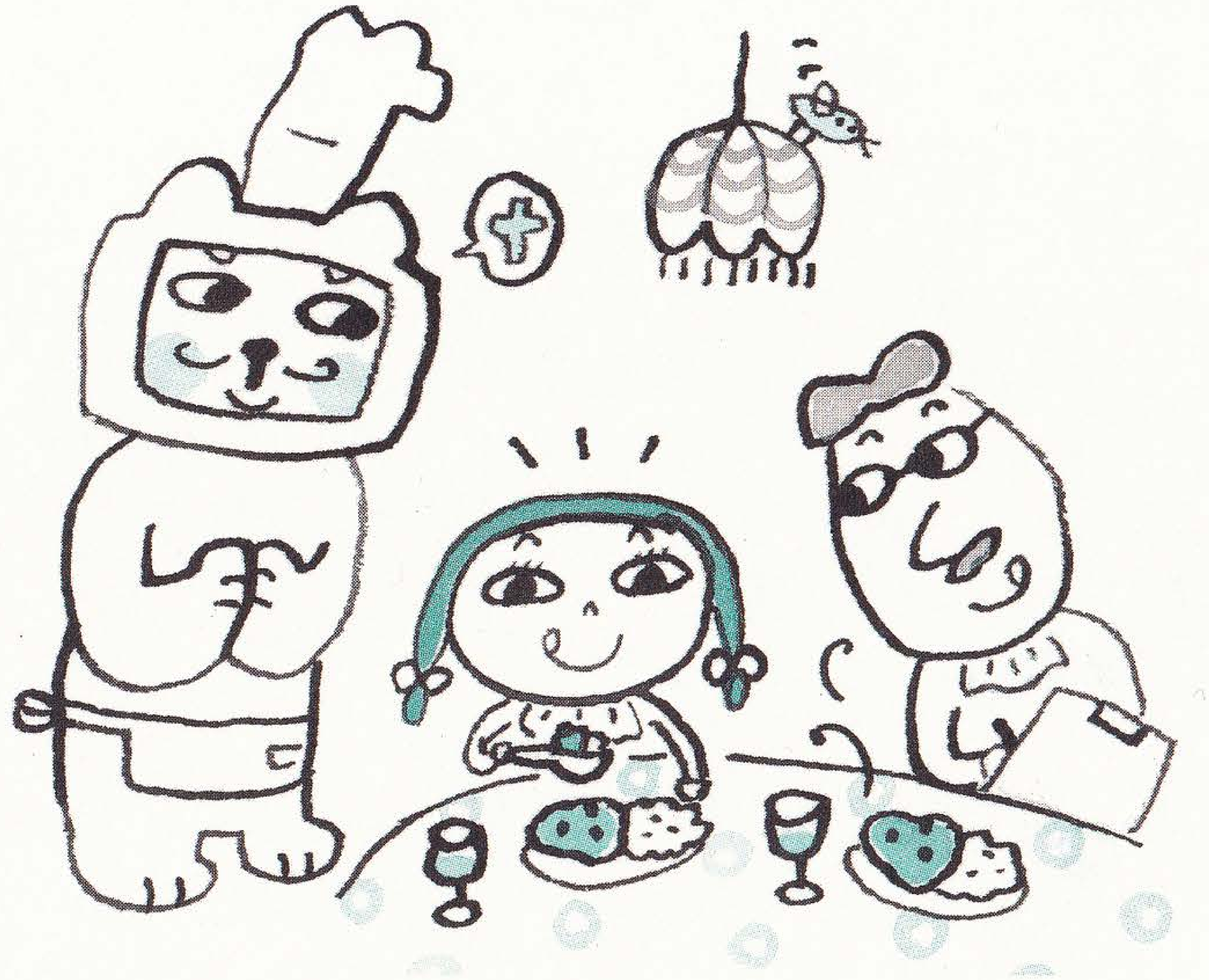
● 設計

次に、作り方を決めたら、
実際に**プログラムを書いて、実行できるようにする**。
この段階で、パソコン上で作成したプログラムを動かせば、
パソコンにして欲しい仕事の通りにパソコンが動くんだ。
この作業を**実装**という。



● 実装

最後に、
動くようになったプログラムは、
きちんと**使えるのかを確認しておく**必要があるんだ。
動いただけでは、まだ実際に使えるかどうか安心できないからね。
この確認の作業を**検査（テスト）**という。



● 検査（テスト）

ここで、一通りプログラミングの全体的な流れをつかむことができたね。

「はい。イメージをつかむことができました。」

このように、プログラミングは全体を通して「分析」「設計」「実装」「検査（テスト）」の流れに沿って行われているんだ。

ポイント

プログラミング作業は、分析、設計、実装、検査を通して行われる

分析

プログラミングの全体的なイメージをつかんだら、この流れに沿って、かんたんなプログラムを作って確認してみることにしよう。

早速、くま君のパソコンで【ようこそ、森の世界へ】とパソコン画面に文字を表示するプログラムを作ってみよう。

この**ようこそ、森の世界へ**を表示するプログラムつまり、
“**どんなプログラムを作るか**”
が、**分析**に当てはまるものだ。

さらに、この仕事の内容を分析してみよう。
このときに役立つのが**5W1H**だ。

5W1Hは、

- who（だれが）
- when（いつ）
- where（どこで）
- why（なぜ）
- what（何を）
- How（どのように）

を表し、私たちが人と話をするときや文章を書くときに基本的な要素として用いられる。

これが、プログラムを作成する際、分析や計画を立てる時にも
重要な考え方になるんだ。
それでは、1つひとつみていこう。

Who（誰が）

誰がこのプログラムを使うのかということを考える。
このプログラムは、
くま君や**絵美**が**使う**んだね。

When（いつ）

どんなときに、
このプログラムを使うのかということを考える。
このプログラムは、
今回、絵美とくま君が
プログラムを実行して、**文字を表示できることを確認する**
といった作業の時に使うんだね。

Where（どこで）

どこでこのプログラムを使うのかということを考えるんだ。
これはくま君のパソコン上で使うよね。
だから、**どこで**の場所に当てはまるのは、**パソコン上**だ。

Why（なぜ）

なぜ、このプログラムを使うのかということだ。
プログラムを作成する**目的**をはっきりさせる。

これは、
絵美とくま君のために、かんたんなプログラムを作成して、
プログラミング全体の流れをよりイメージしやすいようにする目的で、プログラムを作る、ということなんだ。

さらにプログラムを作成した場合、きちんと実行し動くことができるか、文字がきちんと表示されるかどうかを確認するために
このプログラムを作るという目的も含まれる。

What（何を）

このプログラムで何をするのかということだ。
最終的にパソコンの画面に
「ようこそ、森の世界へ」と表示する
というのが答えだね。

この、パソコンの画面に文字を表示させるまでの過程を
「**シナリオ**」とか「**フロー**」というんだ。

シナリオ（フロー）

① 人間がパソコンの電源を入れる

↓

② パソコンが立ち上がって、入力待ちの待機状態になる

↓

③ 人間がプログラムファイルを起動する
（.exe ファイルをダブルクリックする）

↓

④ パソコンの画面上に
「ようこそ、森の世界へ」が表示される

How（どのように）

どのようにこのプログラムが動くのかということだね。
これは、設計で詳しく考えるけれど、
くま君のパソコンは、WindowsというOSで動いているので、
その環境の**パソコン画面上で、文字が表示されるようになる**という
わけだ。

ポイント

- 分析は、どんなプログラムを作成していきたいかを考える作業のことをいう
- 分析は、5W1H（だれが、いつ、どこで、なぜ、何を、どのように）で明確にするとわかりやすい

設計

次に設計をしていこう。

WhatやHowで考えたシナリオを（P86参照）、
“人間がすること”と“パソコンがすること”にわけてみよう。

人間	パソコン
① パソコンの電源を入れる	② プログラムを動かす準備をする
―	③ 入力待ちを表示する
④ プログラムファイルを探す	―
⑤ プログラムを起動する	⑥ プログラムを動かす
―	⑦ 「ようこそ、森の世界へ」を表示する
―	⑧ プログラムを終了する
⑨ 表示を確認する	―

● 設計は分析をさらにわける作業

このように、設計では分析をさらにわけて作業を考えていくんだ。
すると、

・プログラムが動くきっかけ
・プログラムが動くまでの流れ
・動いた後の結果
・終了

が明確になり、
作成したプログラムがパソコンの中でどのように動くのかがわかるようになる。それから、パソコン内部での動きがわかった後、コンピュータでどのように実現していくのかを考えていくんだ（P207参照）。

全体の中で、
何が、**どこ**にあって、**何をするのか**
ということを明確にしていく設計作業は、とても大切なことなんだよ。

ポイント

- 設計は、分析のシナリオから全体的な流れを導き出すことができる
- 設計段階で、「はじめ」〜「終了」までを明確にすることで、プログラムがどのように動き、どのように実現するのかが明確になる

実装①準備とソースコードの作成

プログラムの設計をもとに、
実際にプログラムを書いて実行してみよう。



「お父さん、プログラムに何を書くのかわかったけれど、何に書けばいいの？」

プログラムを書くには「書く」ことのできる、道具が必要になるんだ。
まず必要な道具をそろえることにしよう。



「先生、プログラムを作るのに必要な道具って何？」

プログラムを書くには**エディタ**というものが必要になる。
エディタは、かんたんにいってしまうと、
ワープロソフトのようなもので、いわゆる**原稿用紙**だ。
この原稿用紙にプログラムを書いて、
コンピュータが理解できるようにするんだ。

実際には、コンピュータが直接理解できる、
プログラム（翻訳プログラム）に伝えるためのもので、
このエディタにプログラムを書いておけば、
プログラムの内容を正しく理解して、コンピュータにわかるように伝えてくれるんだ。


「お父さん、このエディタはどうすれば手に入るの？」

このエディタは、**ソフトウェア開発ツール**として、
インターネットからダウンロードすることができるんだ。


「ソフトウェア開発ツール？」

プログラムをつくるのに、必要な道具つまり、エディタ、翻訳プログラムなど、**プログラミングに必要なツール**のことだよ。

インターネットからソフトウェア開発ツールをダウンロードする場合には、検索エンジンで“**ソフト**”とか、“**開発環境**”などをキーワードに探すんだよ。
これには、無償のものから有償のものまであるんだ。
また、使いやすいものもあれば、少し使いにくいものもあるので、これらの中から自分に合ったものを選択してダウンロードしてみよう。

ちなみに基本的なソフトウェアの開発環境には、次のようなものがある。

【無償】

・GNU （gcc、g++）

http://www.gnu.org/home/ja.html

→無償の王道はGNU。Linuxの元になった無償のソフト開発環境代表のソフトウェア。コンピュータやOSの基本知識が必要で参考書を準備して取り組みたい。

・C++Builder

http://www.borland.com/downloads/download_cbuilder.html

→開発ツールの核となる非常に高速な32ビットの最適化ソフトウェア。プログラミング技術を広めるために有償から無償化した、立派な開発環境。これも使いたいソフトウェアのひとつ。

【無償／有償（開発に関わるガイド付きパッケージ）】

・Visual C++ 2005 Express Edition（日本語版）

http://www.microsoft.com/japan/msdn/vstudio/express/visualc/

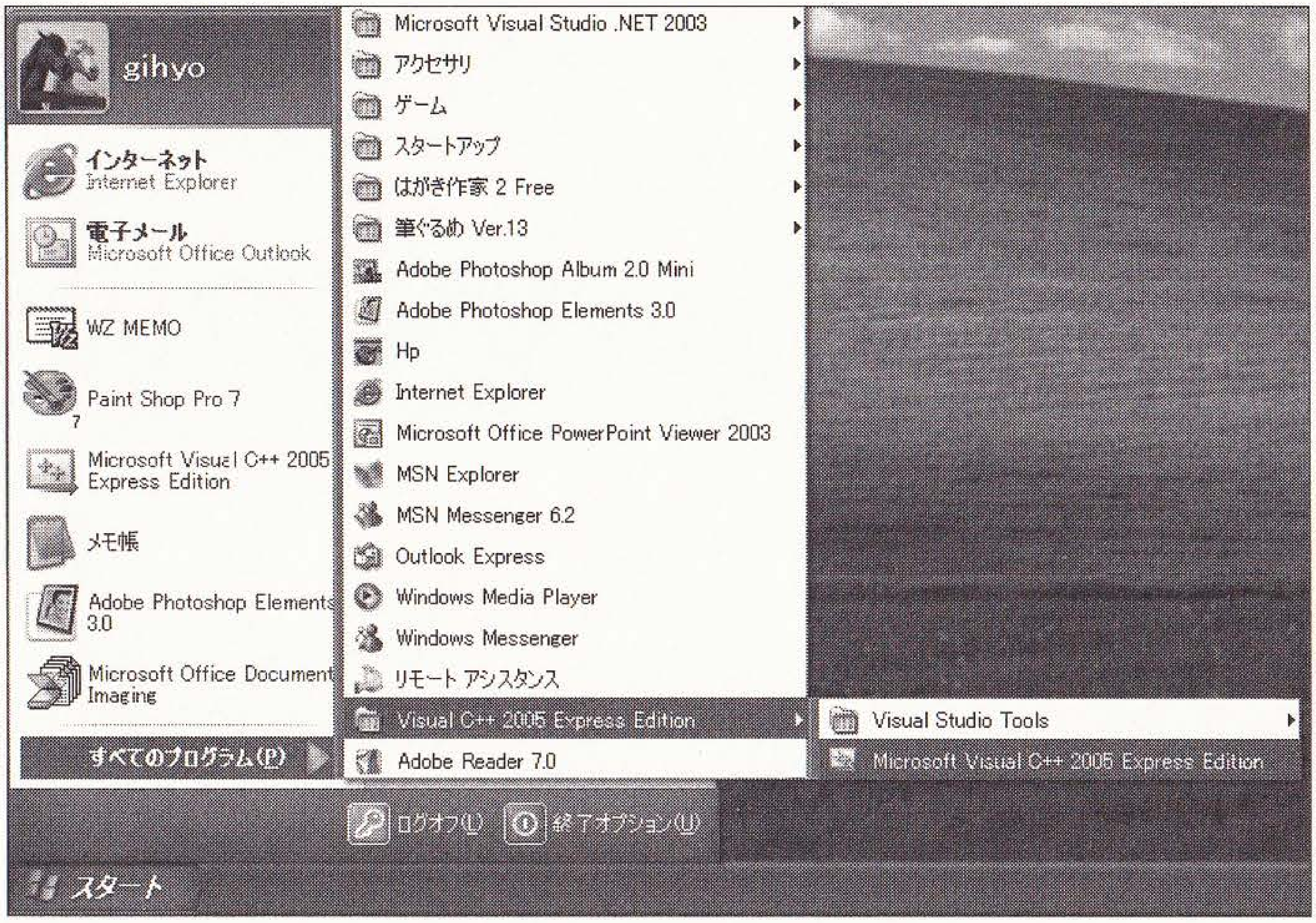
→コンパイラ、コード エディタ、デバッガなどを備えた、強力なソフトウェア。ダウンロード版は無料。Windowsで使う場合、初心者にとっては最も操作しやすい開発環境。ただし、PCには必要なメモリや処理性能が要求される。

ここでは、Microsoft Visual C++を使うことにしよう。

インストールや起動の方法は、Microsoft Visual C++の説明書や参考書、ホームページをみて確認してみよう。

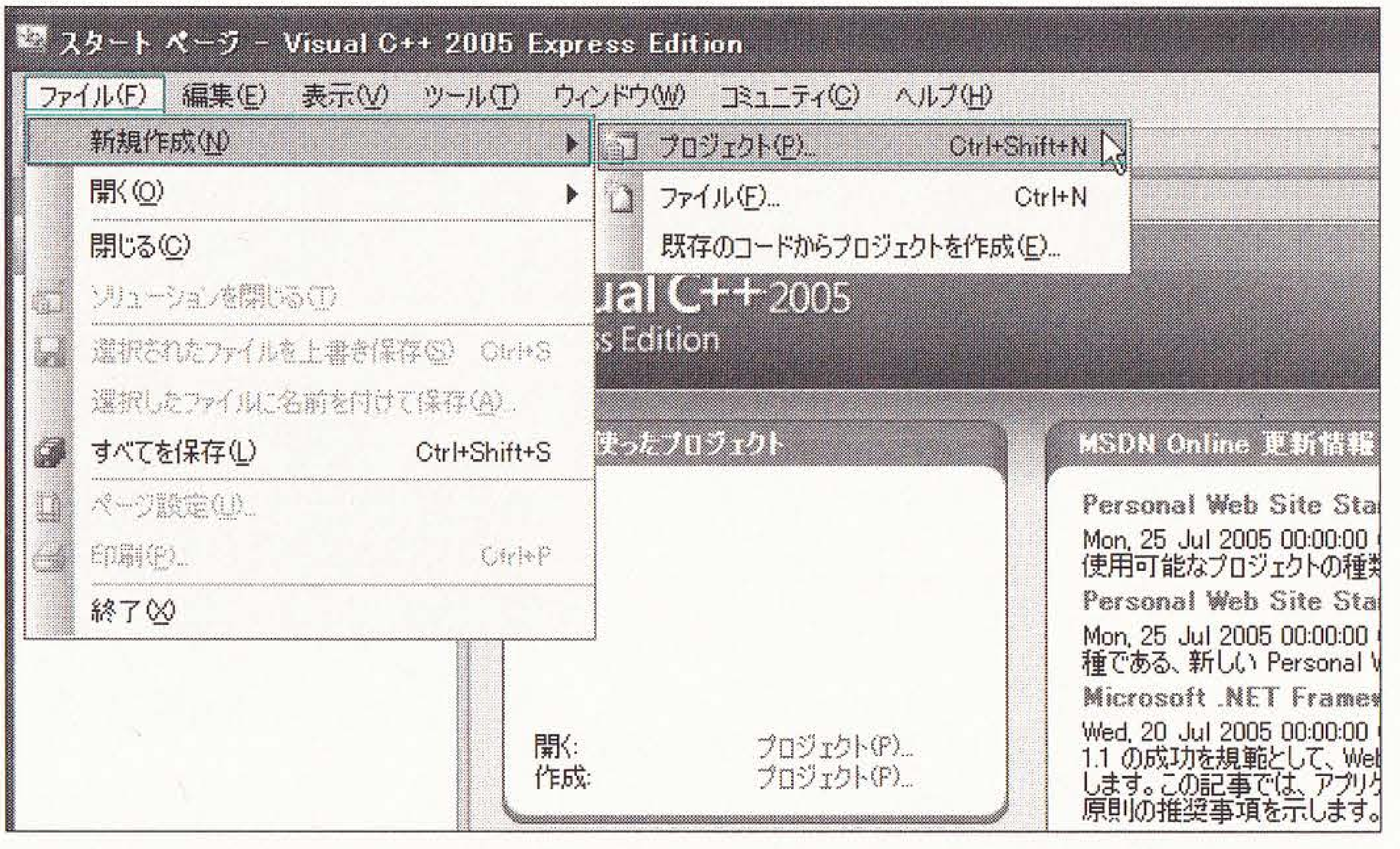
プログラムの作成手順

1. [スタート]-[すべてのプログラム]-[Visual C++ 2005 Express Edition]-[Microsoft Visual C++ 2005 Express Edition]を選択し、プログラムを起動する。



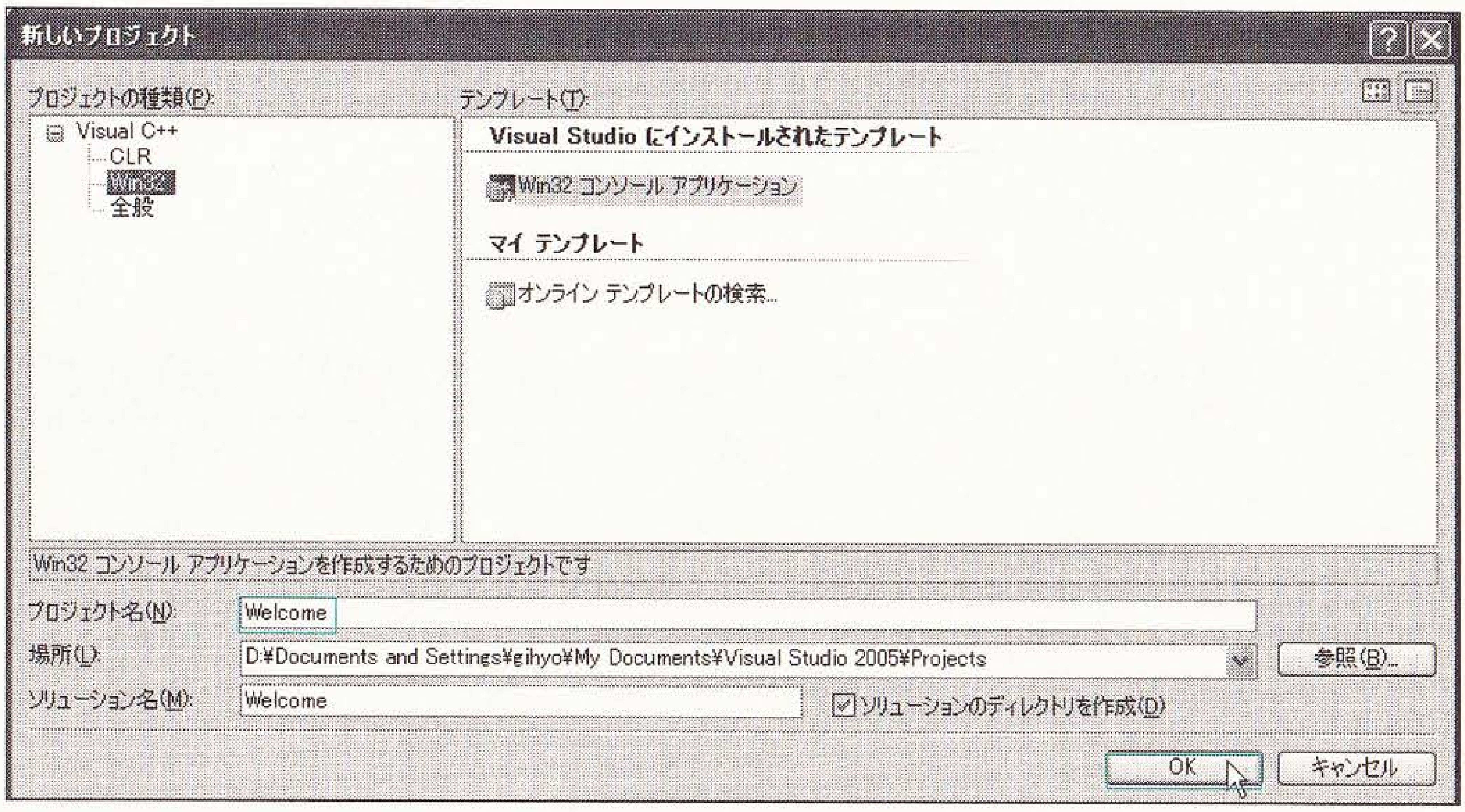
[visual C++2005 Express Edition]の起動

2. [スタートページ-Visual C++ 2005 Express Edition]ウィザード画面が開いたら、メニューから[ファイル]-[新規作成]-[プロジェクト]を選択する。



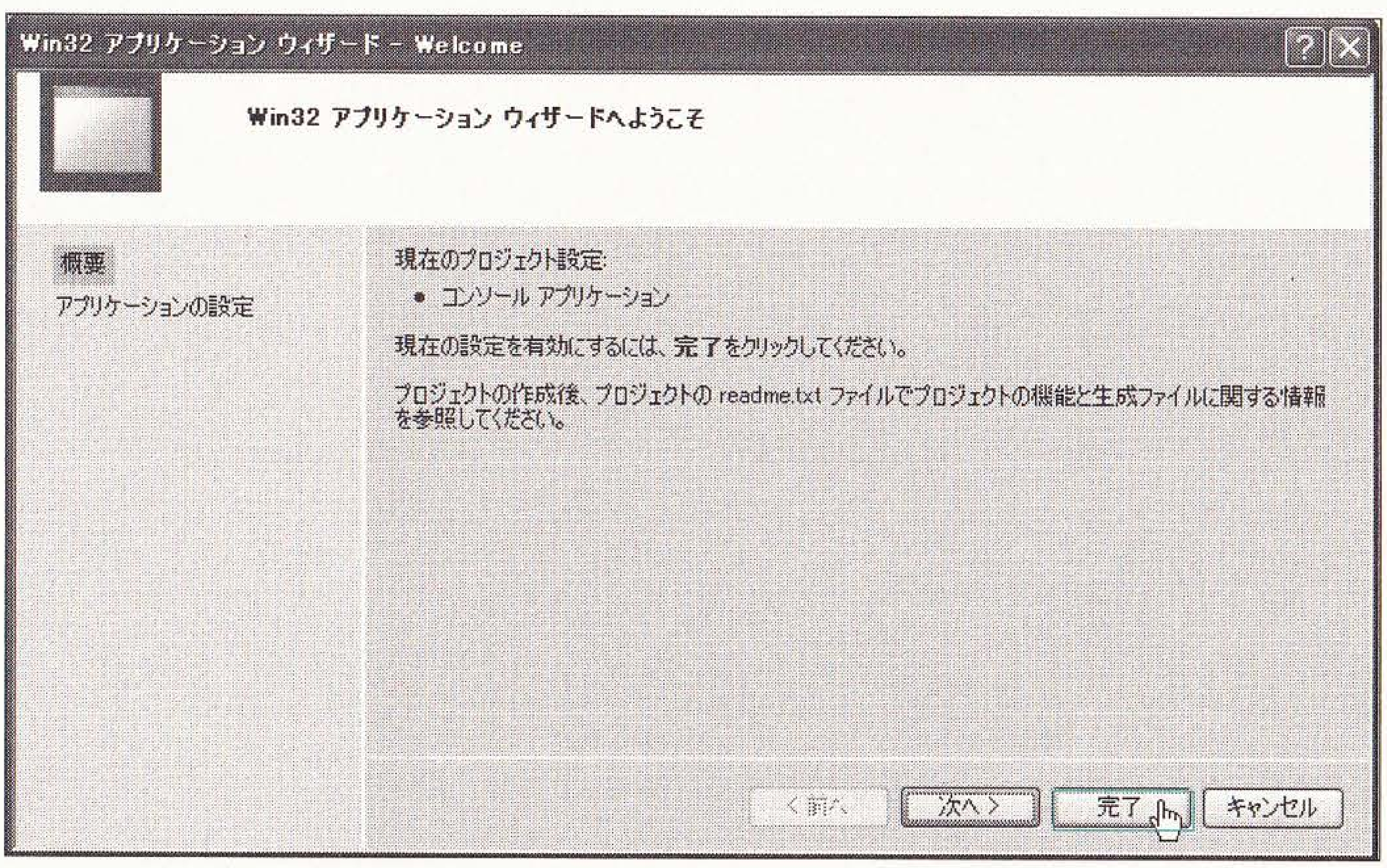
● 新規プロジェクトの選択

3. [新しいプロジェクト]ウィンドウが開く。[プロジェクトの種類]欄で[Win32]を選択し、[プロジェクト名]を「Welcome」にして、[OK]ボタンをクリックする。



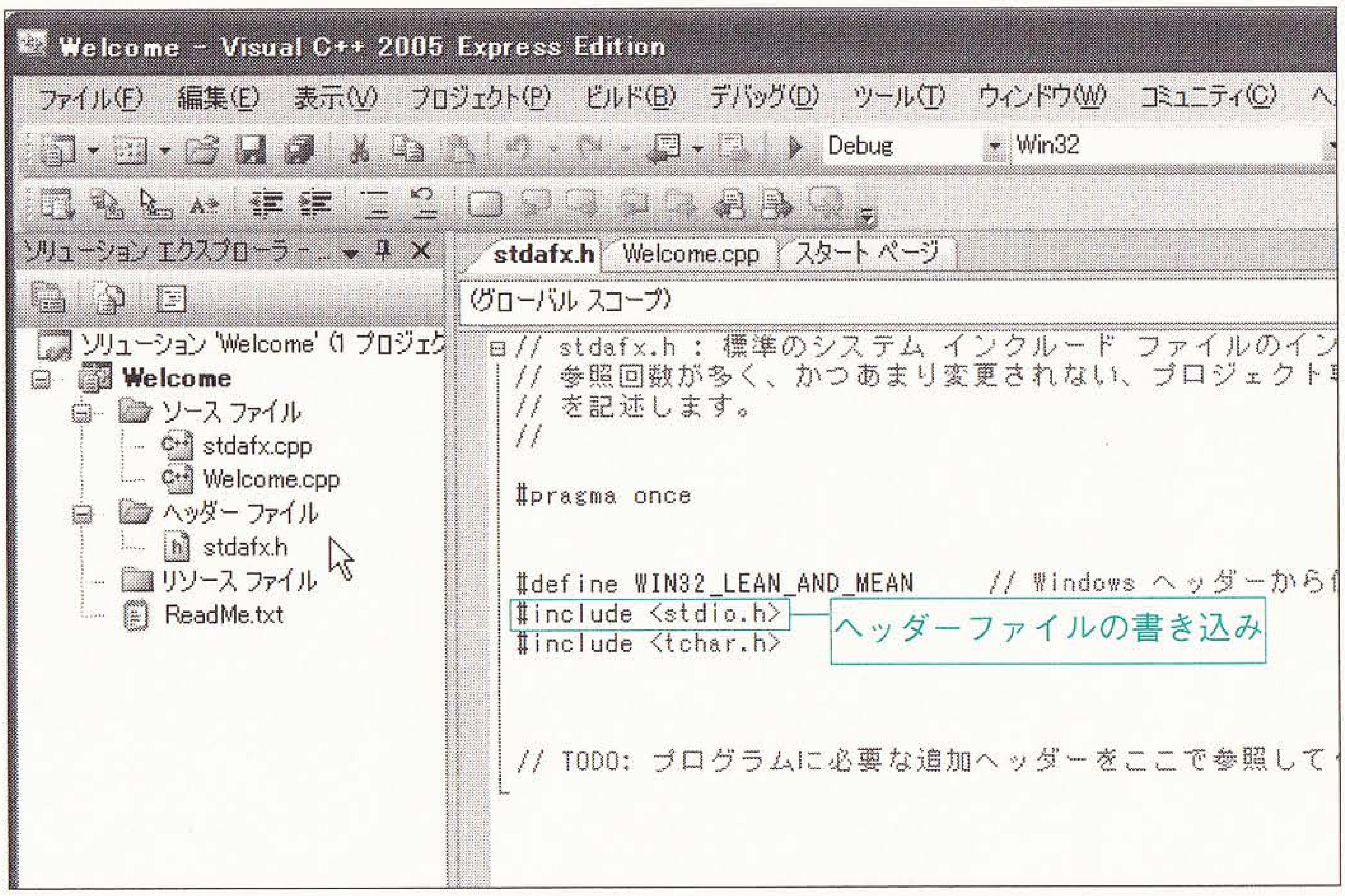
● プロジェクト名の入力

4. [Win32 アプリケーション ウィザード-Welcome]が表示されるので[完了]をクリックする。



● アプリケーションウィザードの完了

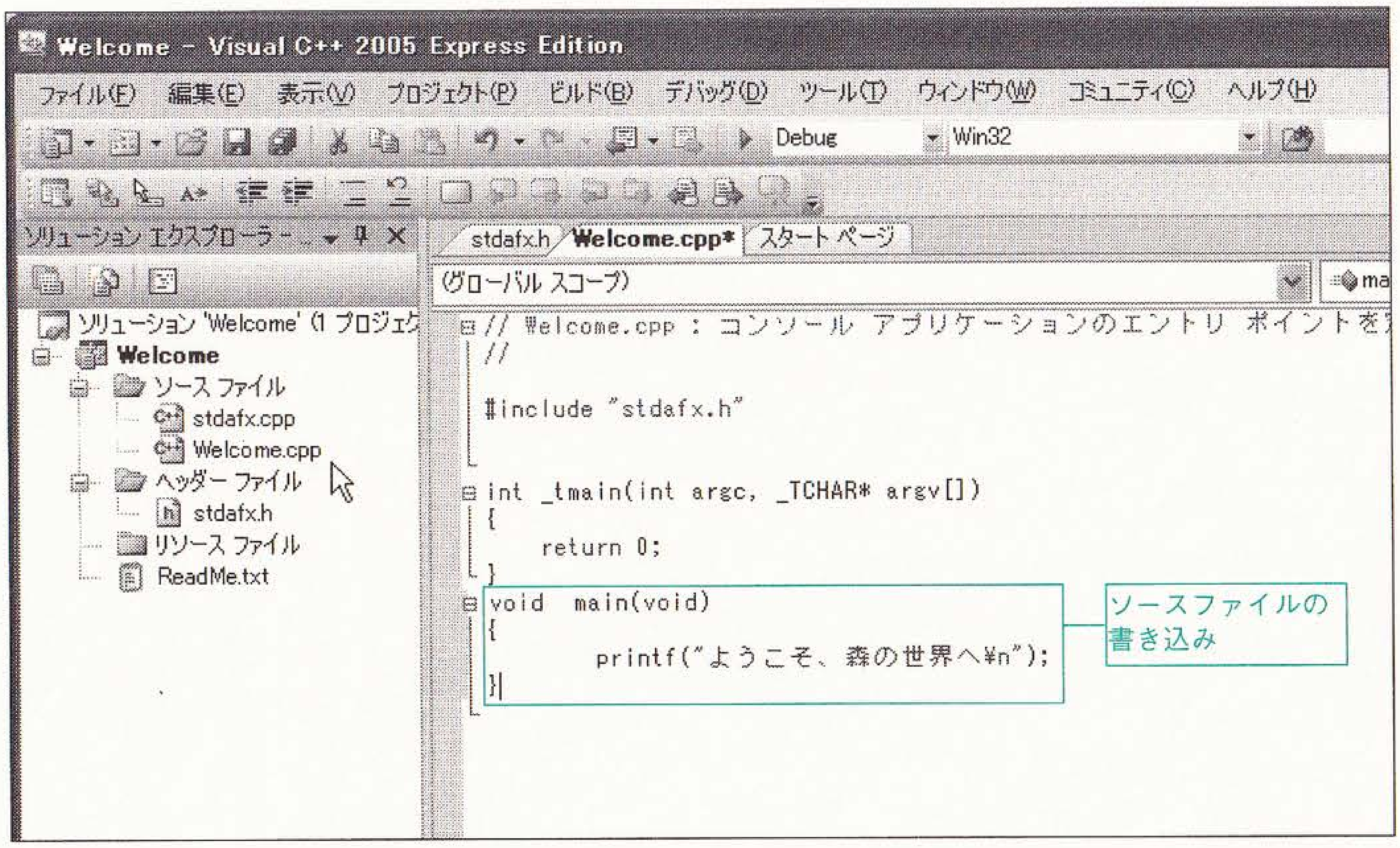
5. リスト3-1（P98参照）からソースコードの書き込みを行う。まず、[ヘッダーファイル]-[stdafx.h]をダブルクリックし、「＃include <stdio.h>」が定義されていることを確認しよう。



● ソースコードの書き込み（ヘッダー）

次に、[ソースファイル]-[Welcome.cpp]をダブルクリックし、次のように、プログラム本体にソースコード(プログラムそのもの)を書き込もう。

```
void main(void)
{
        printf("ようこそ、森の世界へ¥n");
}
```



ソースコードの書き込み（本体部分）

ソースコードをこの翻訳プログラム（P101参照）に翻訳してもらうと、機械語のプログラムができあがるんだ。

つまり、コンピュータはこのソースコードを理解することで、
コードに書かれた通りに働くことができる。
ソースコードというのは、**プログラムそのもの**を指しているので、
命令（**コマンド**）の内容を仕事の順番に書いていく必要がある。

また、ソースコードを記述する時は、
使用する**プログラミング言語の規則に従う**ことが大切だ。



「プログラミング言語の規則って？」

コードを入力するときのきまりのことなんだ。
たとえば、次のようなものがあるので、この規則を守って書くことにしよう。

・英数字は**半角**で入力する
・空白部分は、**スペースキー**または**tabキー**を押す
・セミコロン（;）、コロン（：）、{}、[　]、() の違いに気をつける
・0（ゼロ）、O(オー)、1（数字）、l（エル）の入力に気をつける

実際のプログラムは、次のリスト3－1のようになる。

リスト3-1

```
#include <stdio.h>

void main(void)
{
    printf("ようこそ、森の世界へ¥n");

}
```

表示
改行

では、1つひとつみていこう。
まず、コンピュータの画面に表示をするので、
表示するためのしくみを使うことを宣言する。
これをプログラムで書くと、

```
#include <stdio.h>
```

となる。
これは、**ヘッダー部分**を表す。
次にプログラムの**本体部分**は、次のように表す。

```
void main(void)
```

C言語では、実行する仕事ごとに**main ( ) 関数**というのを定義する。
これがプログラム本体なんだ。
実際、このプログラムで何をするのかというのは、
{ } でくくった中に書くことになるよ。
("ようこそ、森の世界へ¥n") ; に書いてある通り、「ようこそ、森の世界へ」の文字列を表示するということになる。

その次の**¥n**というのは、**改行する**という意味なんだ。
つまり、コンピュータに対して、「ようこそ、森の世界へ」と表示した後で、改行を命令していることになる。
また「ようこそ、森の世界へ」の前にあるprintf( )は、

画面に出力するためのプログラムなんだ。

つまり、ここではstdio.hというファイルに書かれているprintf( )というプログラムで、文字列の出力を行っているということだ。
たった5行だけど、これでも立派なプログラムなんだよ。



「すごい、プログラムができた！」

このようにプログラムは、記述するときの規則に従って、
ソースコードを正確に順番に書いていくことが大切なんだ。

ポイント

- プログラムを作るためには、ソフトウェア開発ツールが必要である。
  このソフトは無償のものから有償のものまである
- ソースコードを書くときには、プログラミング言語の規則に従って記述していく

実装②コンパイルと実行ファイルの作成

プログラムの元になるソースコードの作成ができたね（P97参照）。ここでは、コンピュータがこのソースコードからプログラムを実行する方法について説明していこう。

さて、プログラムを実行するには、**コンパイル**が必要になるんだ。

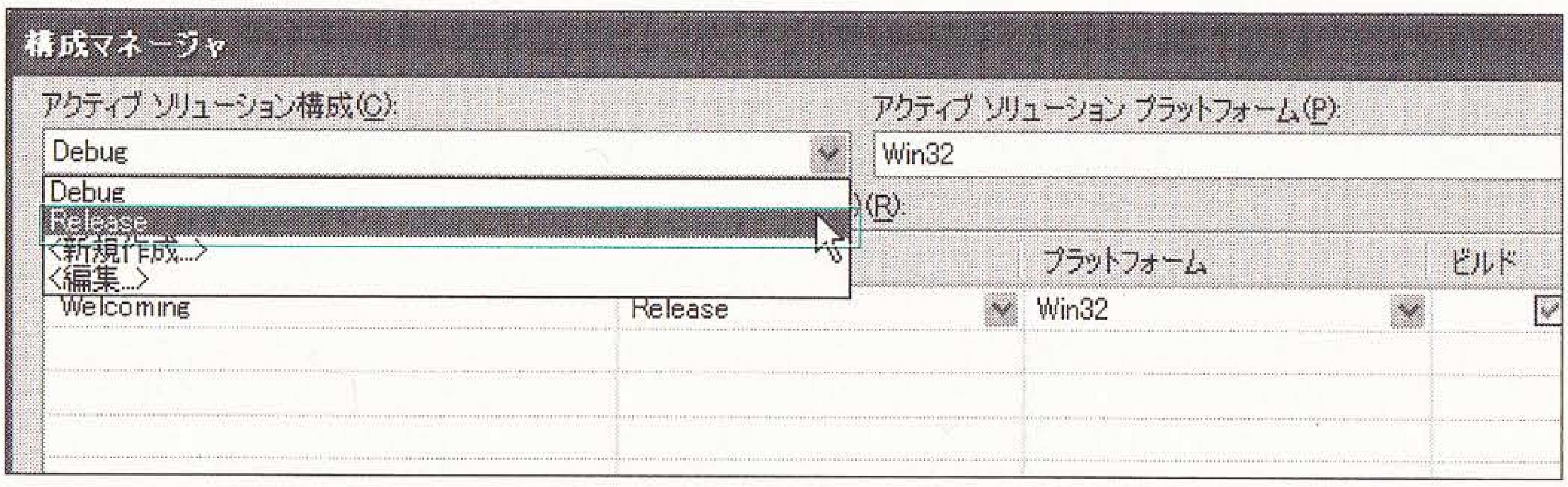
「コンパイル？」

コンパイルするとは、コンパイラと呼ばれる、ソースコードを翻訳して、マシン語にするプログラムで、**翻訳処理を行う**ことをいう。

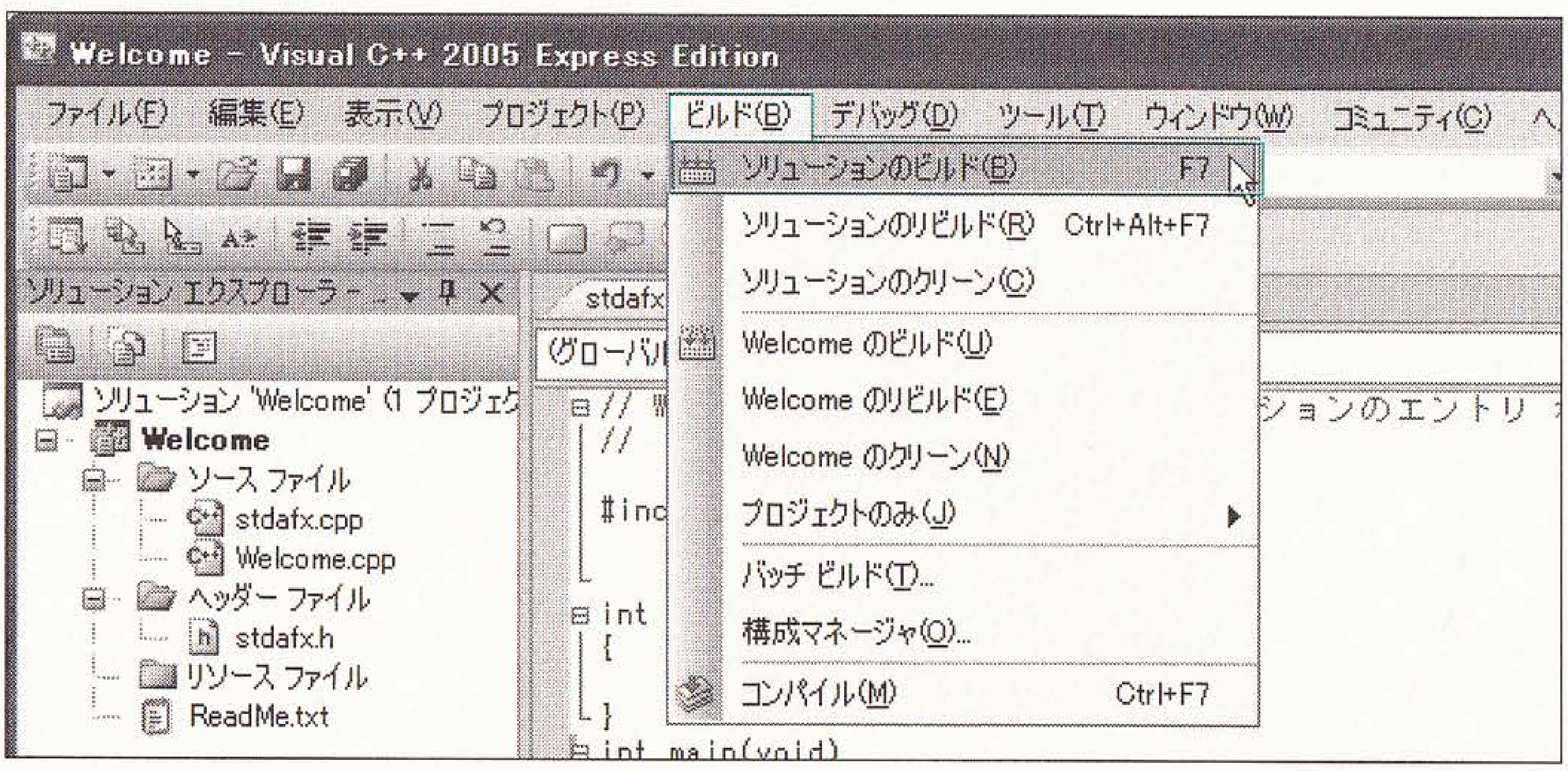
このプログラムを動かすと、指定したファイルを開いて、1行目から順番にソースコードを調べては、翻訳するといった作業を最後の行まで繰り返し行うんだ。翻訳したソースコードは、コンピュータで実行できるマシンコードになって、新たなファイルに書き込まれる。これが、実行可能なファイルになる。
実行可能なファイルを“**オブジェクトファイル**”とか、“**実行ファイル**”、“**Executeファイル**”などというんだ。

「ようこそ、森の世界へ」と表示するプログラムをコンパイルしてみよう。

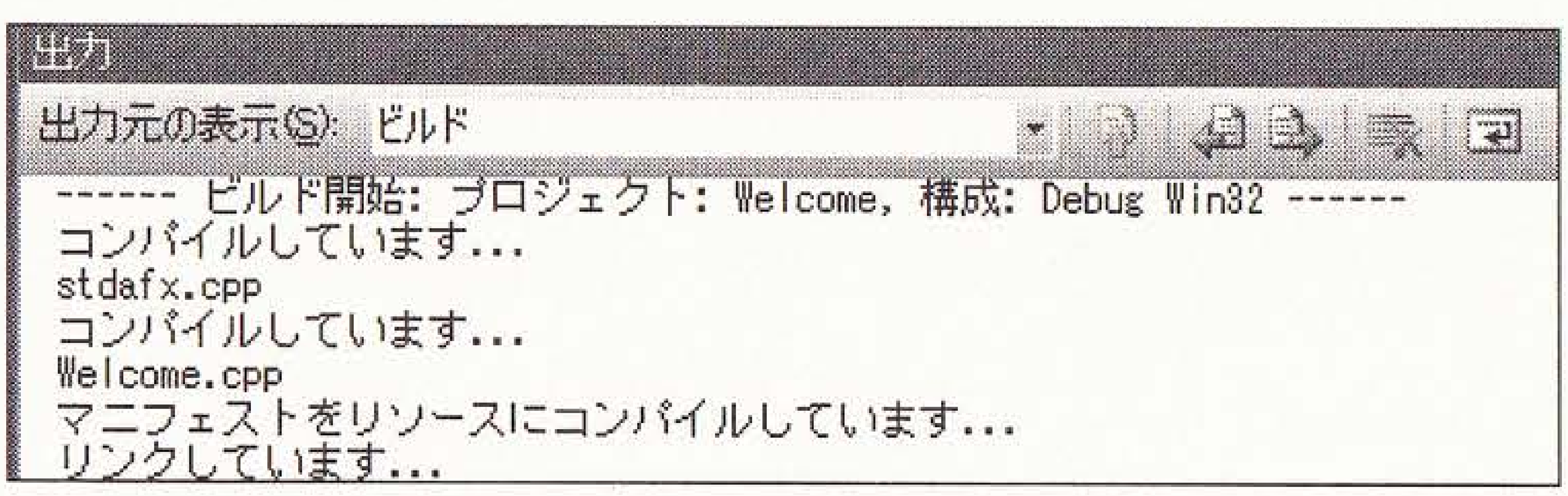
6. メニューから[ビルド]-[構成マネージャ]を開き、[アクティブソリューション構成]を「Release」にして閉じる。次に、メニューから[ビルド]-[ソリューションのビルド]を選択する。これで、ソースファイルの保存が行われ、コンパイルからリンクが実行される。



● [構成マネージャ] を [Release] に

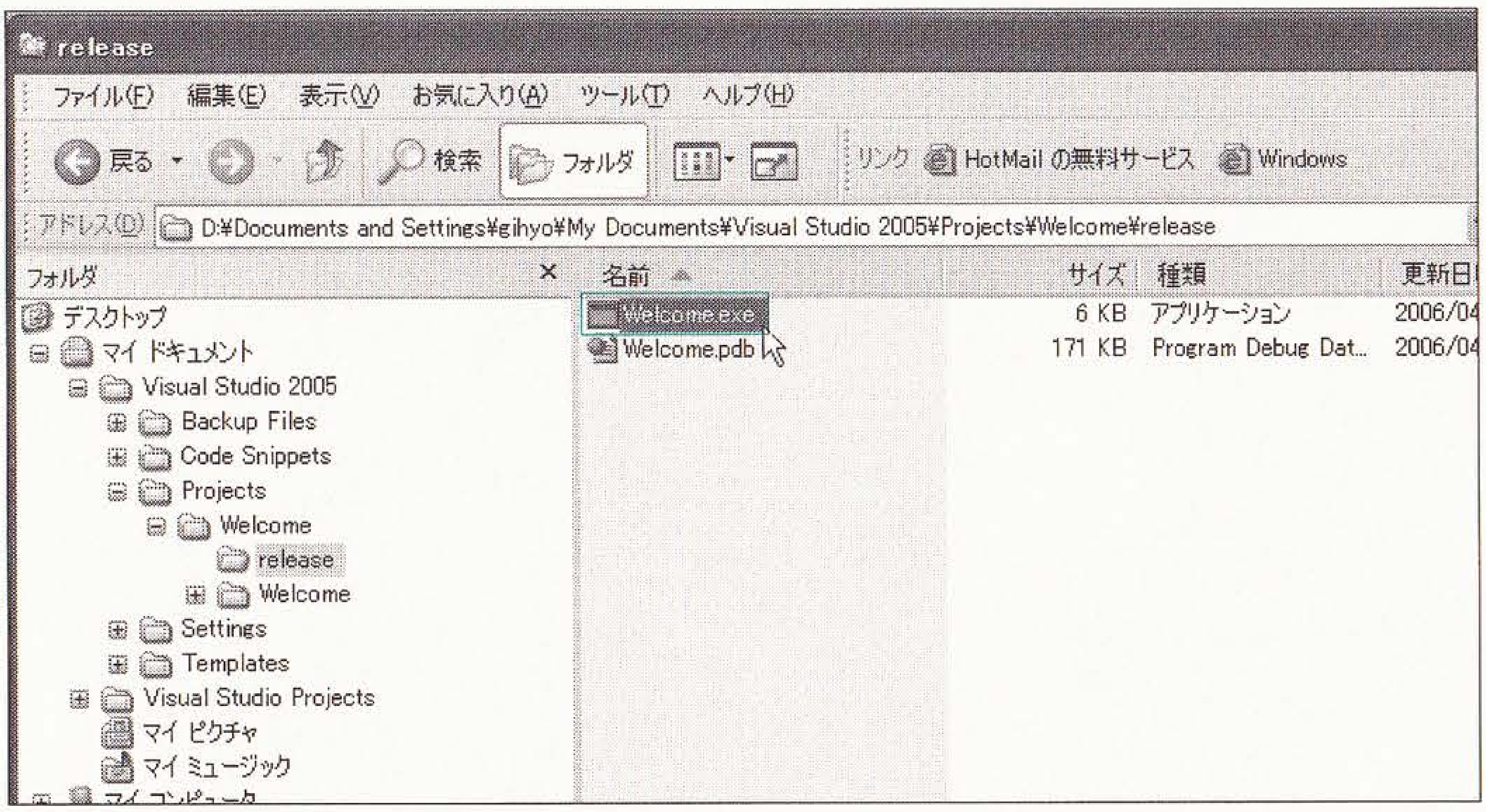


● コンパイルからリンクの実行



● コンパイルとリンクの状況

実行ファイルは、「Visual Stadio 2005」をインストールしたフォルダ内の¥Projects¥Welcomeにreleaseフォルダができあがり、この中にwelcome.exeファイルが保存される。
たとえば、D:¥Documents and Settings¥gihyo¥My Documentsの下にインストールした場合は、
D:¥Documents and Settings¥gihyo¥My Documents
¥Visual Studio 2005¥Projects¥Welcome¥releaseフォルダ内に保存され、このwelcome.exeファイルが実行可能なプログラムとなる。



● 実行ファイル

コンピュータはこのファイルを実行することで、ソースコードに書かれた通りの作業を行うことができるようになるんだ。

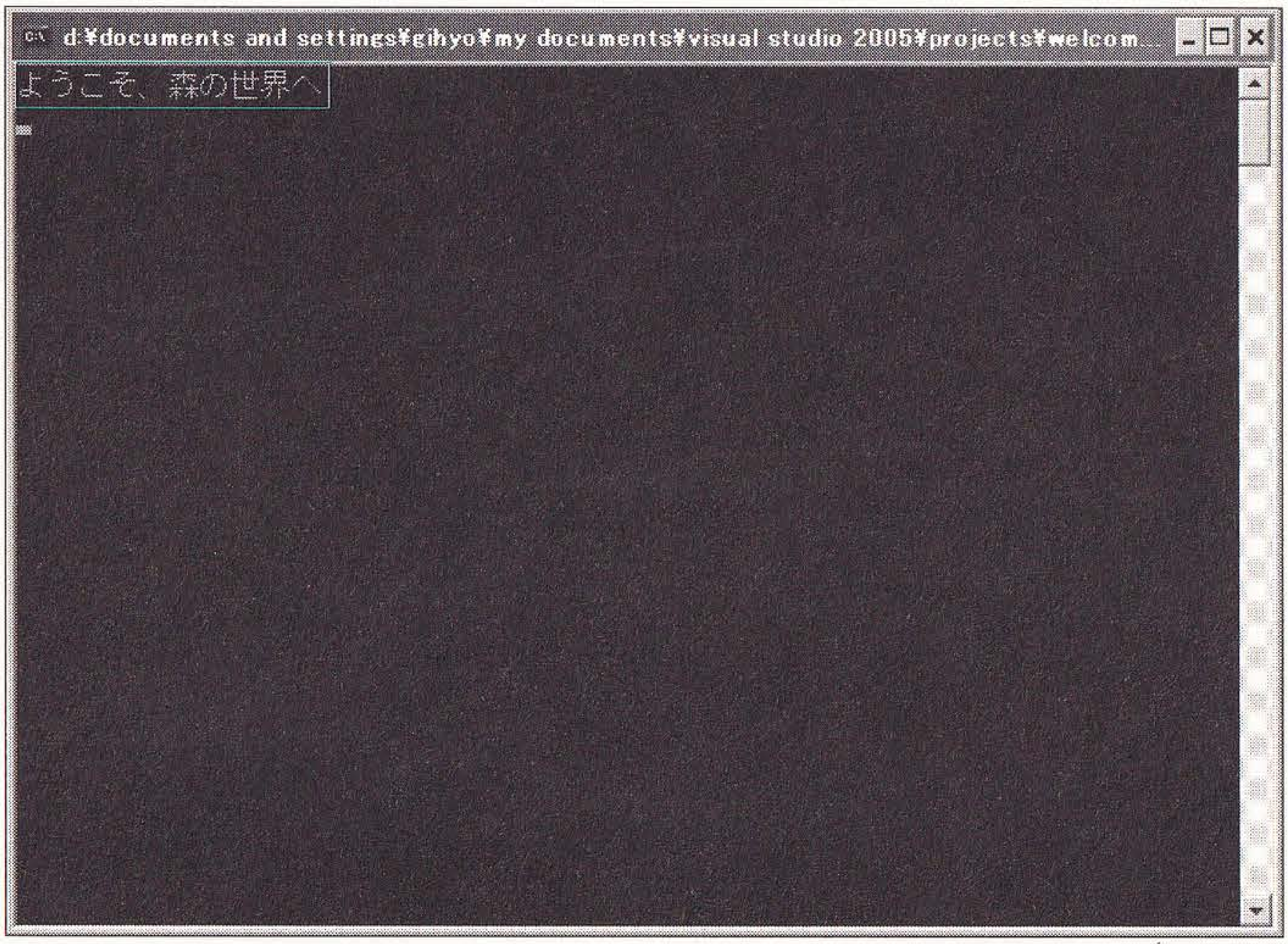
ポイント

- コンパイルは、ソースコードを翻訳しマシン語にすることをいう
- 翻訳後のソースコードは、実行可能なファイルになる

実装③プログラムの実行

コンパイルが行われたら、プログラムを実行してみよう。先にできたD:¥DocumentsandSettings¥gihyo¥MyDocuments¥VisualStudio 2005¥Projects¥Welcome¥releaseフォルダのWelcome.exeをダブルクリックすることでプログラムが実行される。

この時、一瞬、「ようこそ、森の世界へ」と表示されているウィンドウがみえるね。



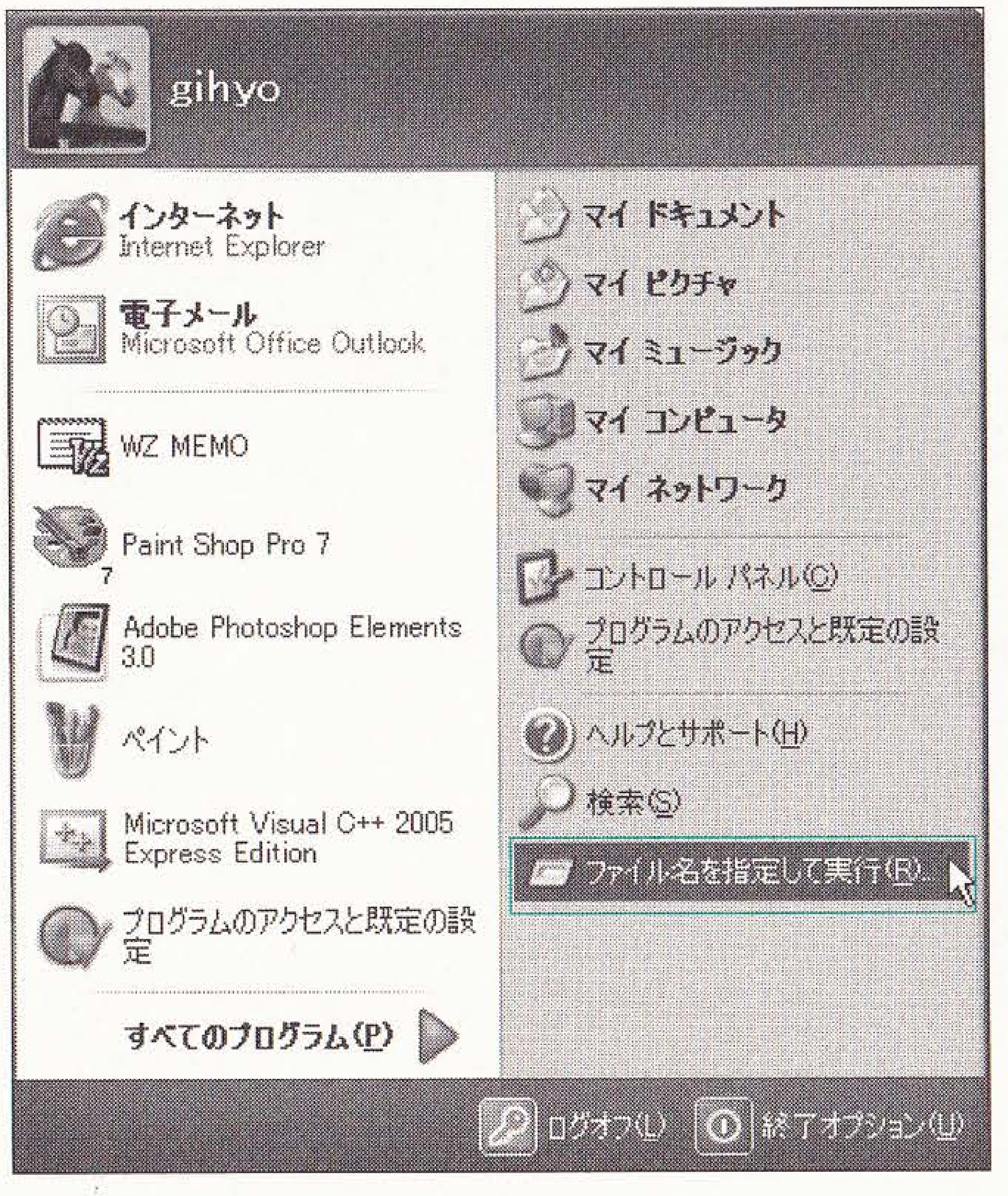
プログラムの実行

「おー、ほんの一瞬ですね」

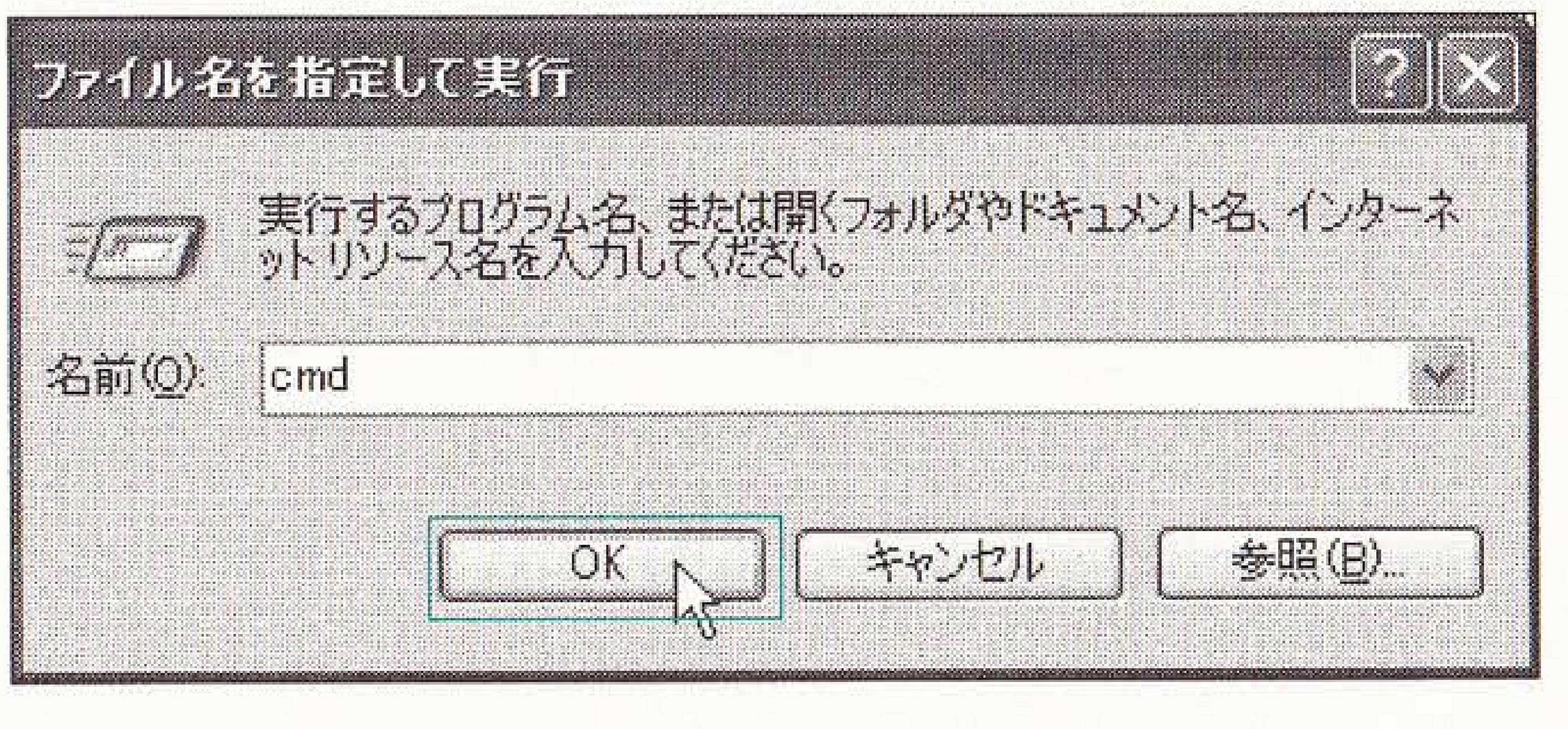
そうなんだ。Windowsでは、プログラムが終了すると同時にお片づけをするプログラムが働いて、せっかく表示した「ようこそ、森の世界へ」という表示をすぐに消してしまうんだ。

そこで、きちんと「ようこそ、森の世界へ」と書かれているのかを確認するため**DOS窓（コマンドプロンプト）**で確認してみよう。

7. デスクトップ画面から[スタート]-[ファイル名を指定して実行]を選択し、[名前]にcmdと打ち込んで、[OK]をクリックする。

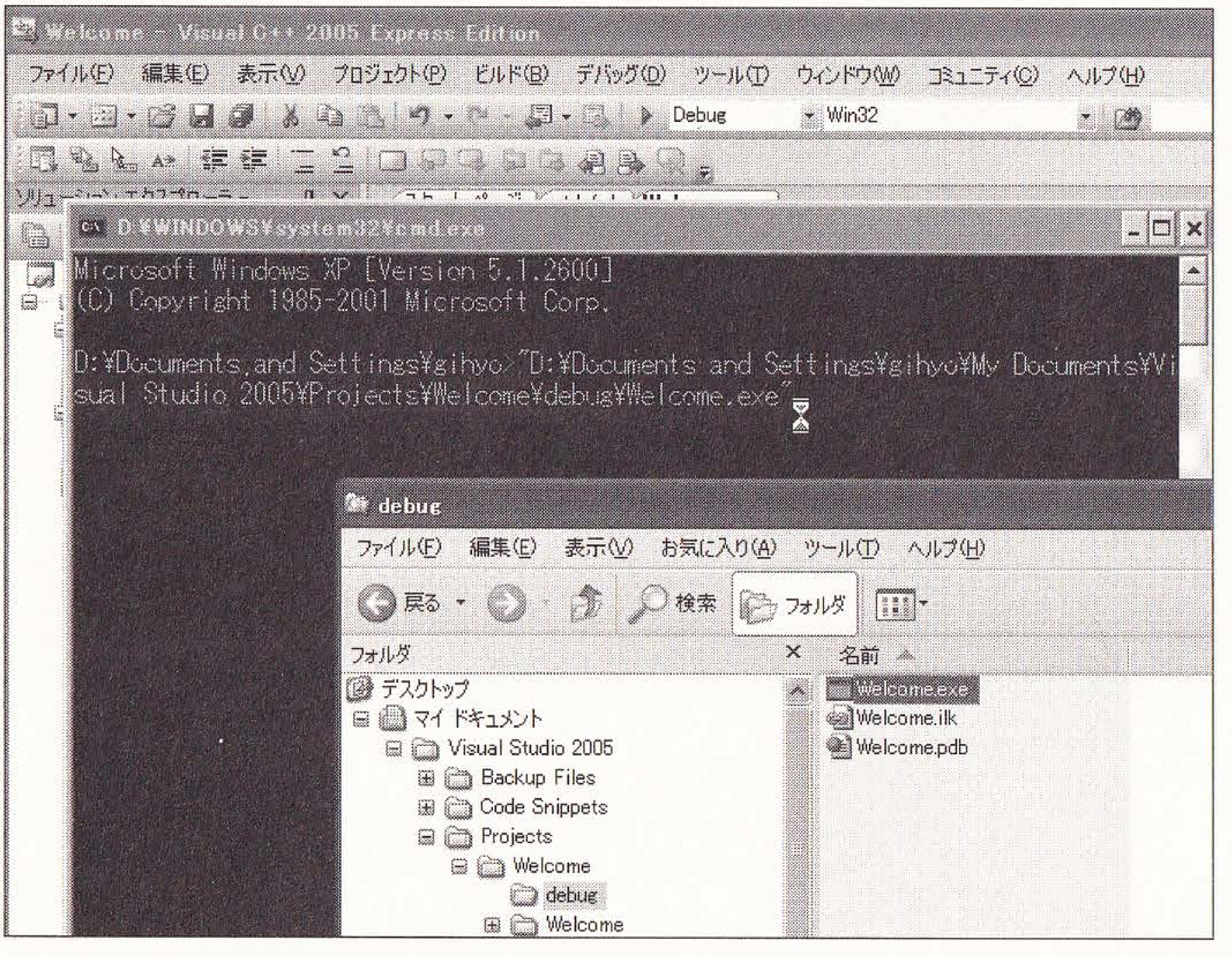


ファイル名を指定して実行（1）



ファイル名を指定して実行（2）

8. DOS窓に、エクスプローラからwelcome.exeをコピーする（ドラッグ&ドロップ）。

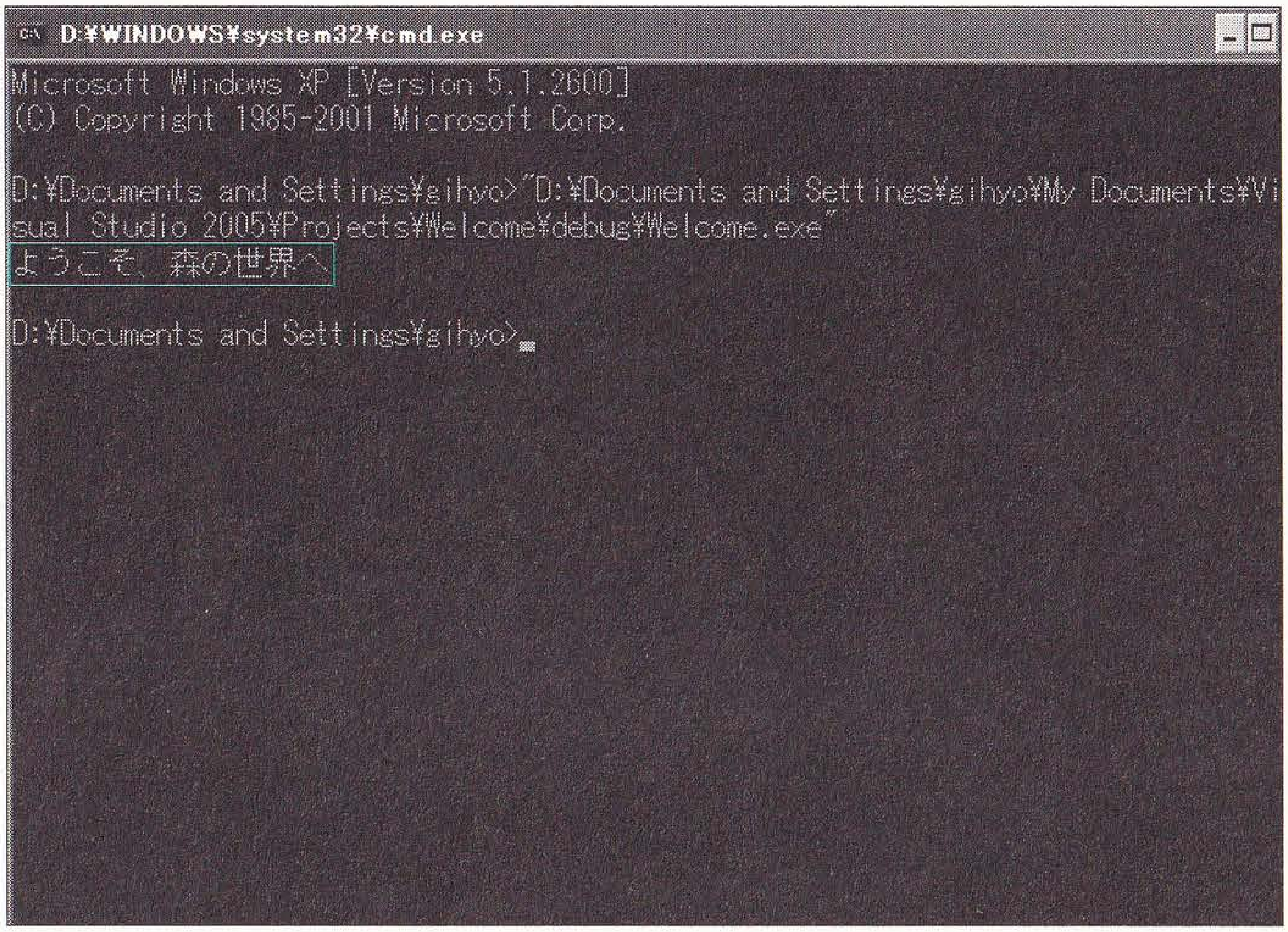


● 実行ファイルをDOS窓にドラッグ&ドロップ

DOS窓にWelcome.exeファイルのある場所(パスという)が表示された。この表示の最後にカーソルが出るようにして、Enterキーを押すと、DOS窓の中でwelcome.exeが実行し、「ようこそ森の世界へ」という文字列が表示される。



「おー。きちんと表示されているね。」



● プログラムの実行（文字列の表示）

さて、DOS窓上では「ようこそ、森の世界へ」の表示がされたことが確認できた。

これまでの作業でプログラミング言語で書いたソースコードをコンパイルすることで、マシンコードができあがり、それを実行することでコンピュータへの命令、つまり「ようこそ、森の世界へ」の文字列を表示することが確認できた、ということになる。

ポイント

オブジェクトファイル（実行可能なファイル）を実行することで、コンピュータへの命令を実行することができる

検査（テスト）

プログラムを実行して、上手く動けば、プログラミングはおしまいなんだけど、プログラミング作業は人がやることだから、間違えることがある。ソースコードを書くときに間違えてしまうと、どんなに小さな間違いでも、コンピュータはきちんと仕事をしてくれなくなってしまうんだ。

ソースコードに入り込んだ、このような間違いを**バグ**という。

バグというのは、英語のBUGで、**虫**のことだ。その昔、コンピュータの中に蛾が飛び込んで、コンピュータが動かなくなったんだけど、この蛾を取り出したら、きちんと動くようになった。このことから、コンピュータの中に虫が入ったようにうまく動かなくなることをバグと呼び、この**蛾を取り除く作業のことをデバッグ**というんだ。

つまり、実行できるプログラムが完成した後に
プログラムが正しく動くかどうか確認する作業が必要になる。

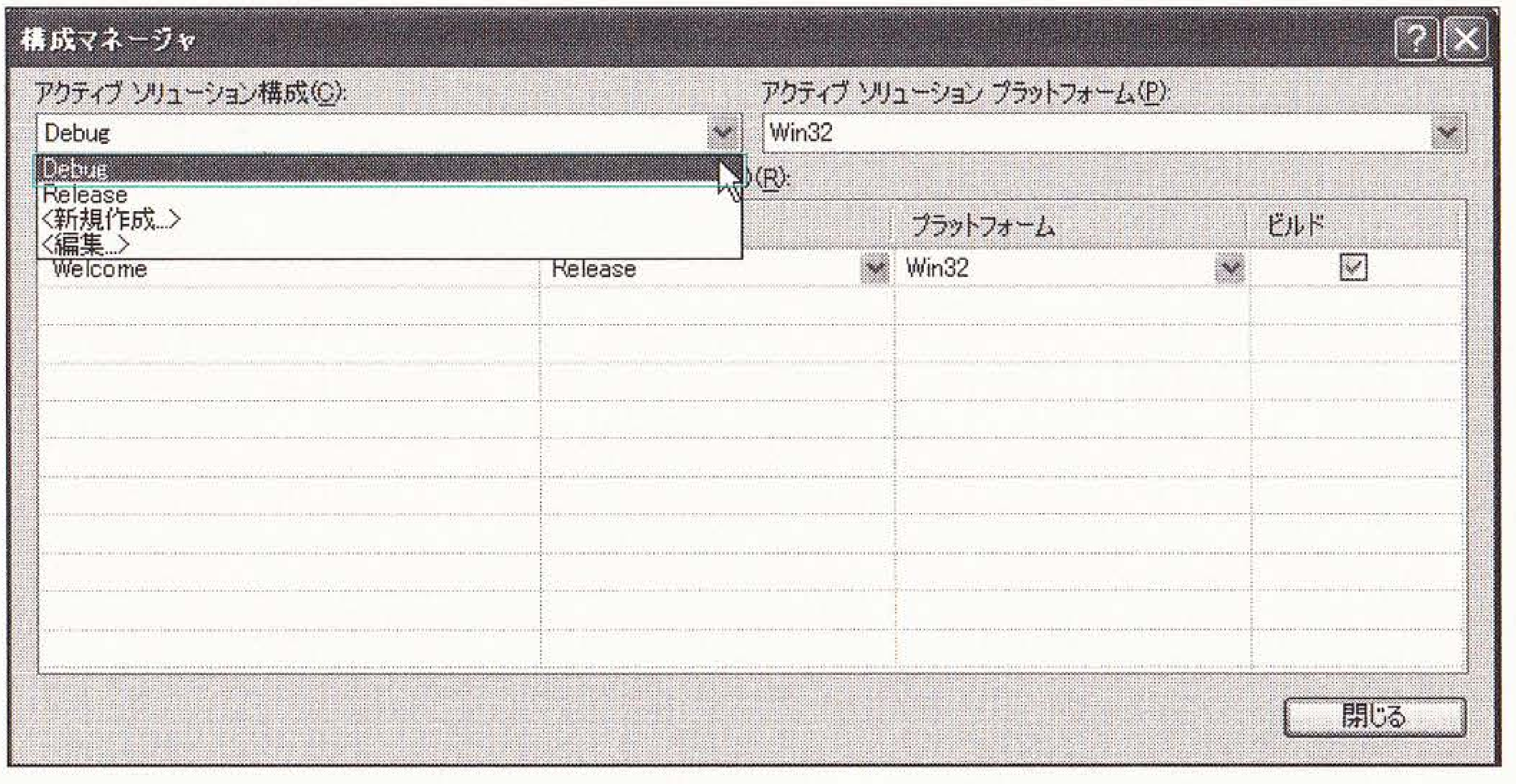
 **「実際にプログラムにバグがみつかったらどうするの？」**

デバッグ作業をしていて、プログラムがうまく動かなかった場合には、うまく動くようにプログラムを修正する必要がある。

修正が終わったら、そのプログラムをもう一度確認する。その結果、また動かなかったらもう一度修正する。そして、さらに確認…というように、プログラムが正しく動くまで繰り返すんだ。
きちんと確認して、正しく動くプログラムにしておかないと後で何が起こるかわからないから、**デバッグ**はとても重要な作業なんだ。

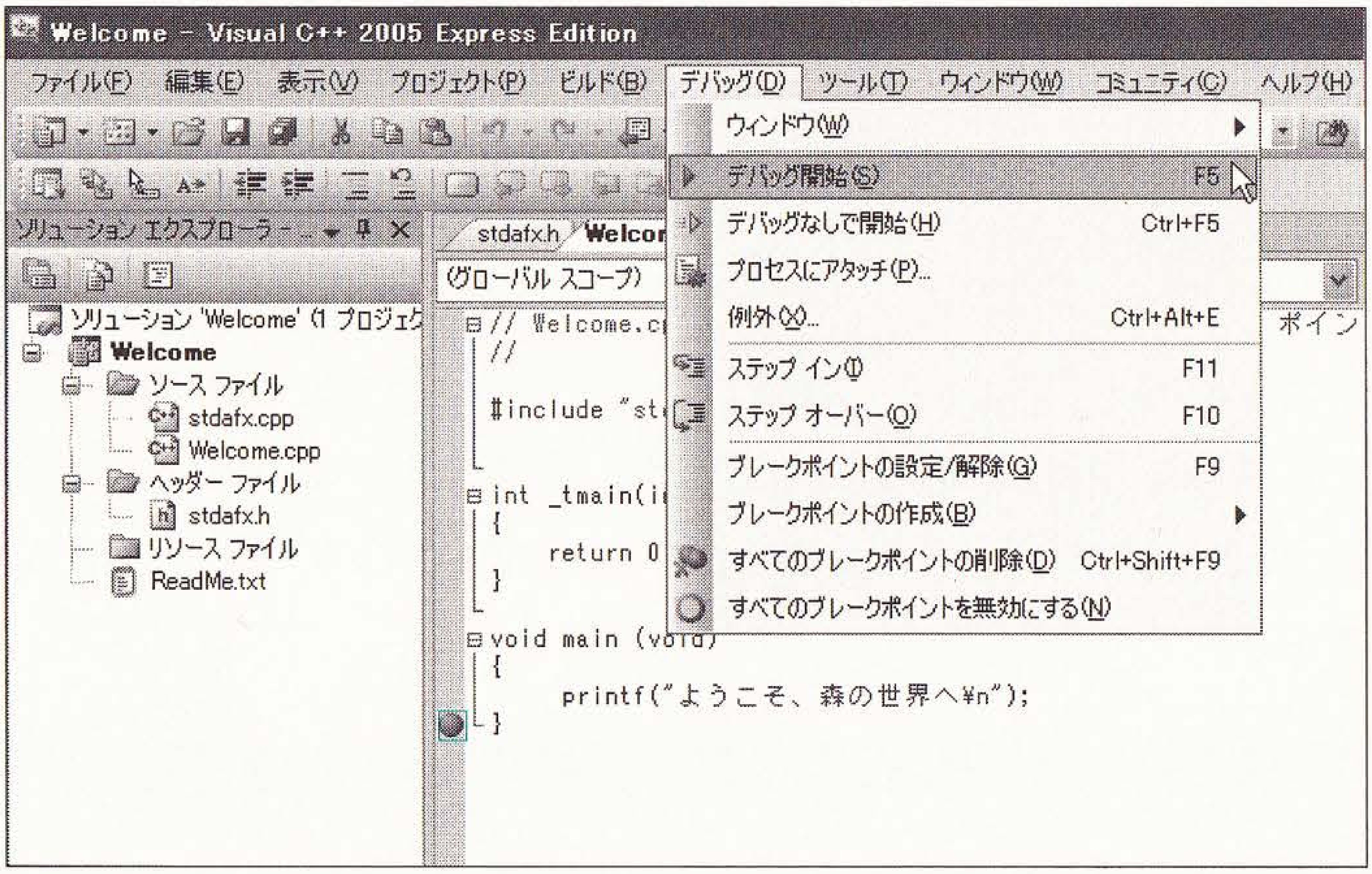
実際のデバッグの作業には、**デバッガー**を使う。
プログラムを開発するツールには、通常デバッガーが備えられていて、プログラミングを行いながら、**エラーを修正する**ことができる。
また、ここでは**ブレークポイント**という機能を用いて、処理を中止し、表示に間違いがないかどうか確認していこう。

9. メニューから[ビルド]-[構成マネージャ]を開いて、[アクティブソリューション構成]を「Debug」にし、閉じる。次にメニューから[ビルド]-[ソリューションのビルド]を選択して、デバッグの準備をする。



[構成マネージャ] を [Debug] に

10. ソースコードの最後（関数が終了するところ）の、}の末尾にカーソルを合わせ、F9キーを押す。
11. F5キーを押し、デバッグ作業を開始する（メニューから[デバッグ]-[デバッグ開始]をクリック）。すると、[コマンドプロンプト]が表示され、文字列を確認することができる（解除する場合は、shift+F5キーを押す）。



● ブレークポイントを用いたデバッグ

これらのデバッグ作業を省いた場合、どうなるかを考えてみよう。
たとえば、「ようこそ、森の世界へ」と表示するプログラムのmain ( )関数の中の記述に、

```
printf("ようこそ、森の世界へ¥n");
```

というのがある。この時、nをmと打ち間違えたとする。つまり、

```
printf("ようこそ、森の世界へ¥m");
```

としてしまうと、

「ようこそ、森の世界へ」

と表示するはずが、

「ようこそ、森の世界へm」

と表示されてしまうことになり、誤った文字列のプログラムができてしまうんだ。
これは、表示した後に改行するための**¥n**が、文字としてのmと認識してしまい、コンパイラでは間違った認識で解釈し、翻訳されたことを示している。その結果、改行されずに**m**という字が表示されてしまうんだ。

しかも、この場合は、文字自体の間違いなのでコンパイルはできて、実行ファイルも作ることができてしまうんだ。

また、このように表示するプログラムの場合は間違いを表示するので、間違ったことを簡単に知ることができるけれど、たとえば冷蔵庫を制御するプログラムのように、処理した結果を表示しないようなプログラムの場合には、間違いに気づくことが難しくなるので注意が必要だ。このようにデバッグ作業は正しいプログラムを作るのに、とても大切な作業なんだ。

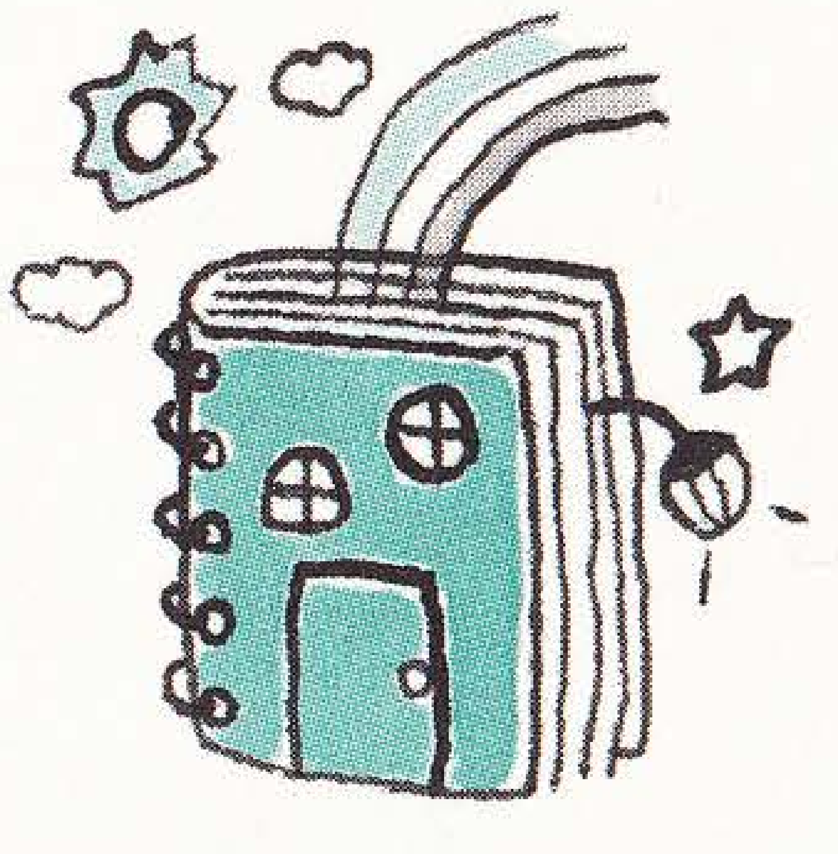
ポイント

- ソースコードの間違いをバグという
- プログラムを作成したら、検査を行い、バグがないかを確認する必要がある。この作業までをプログラミングという

まとめ

- プログラムには、C言語やJavaなどの高級言語が使われる
- プログラミング作業は、分析、設計、実装、検査がある
- 分析は、どのようなプログラムを作成するかを考える作業である
- 設計は、分析をさらに明確にしたもので、「はじめ」～「終わり」までの作業を導くものである
- 実装は、エディタにソースコードを記述して、実行ファイルを作成することをいう
- テスト（検査）は、プログラムを実行して、間違いを発見することをいう

第4章
プログラミング手法



絵美は、お買い物をした時にお小遣い帳をつけているが、
どうやら計算間違いをするらしい。

絵美は、どうしたら間違いなく計算できるか、悩んでいる。

そんなとき、お父さんが絵美には内緒で
仕事の合間に、簡単な電卓（算数）プログラムを作ってみた。

くま「これは何？　先生…。」

あ、見つかってしまった。
くま君、絵美には内緒だからね。

くま「えっ、なに～!!」

シー、静かにしてくれ。

絵美「お父さんとくま君、そこで何コソコソしているの？
**　　　コソコソ話…？　あれ、何それ、お父さん…。」**

どうやら、みつかってしまったようだ…。

これは計算間違いをしない、「プログラム」と呼ばれるもので、
いわゆる電卓と呼ばれるものだ。

絵美「いいな～、それ貸して欲しいよ、お父さん。お小遣いの計算が
**　　　なかなか上手にできないの。」**

電卓を貸すのはいいけれど、
絵美はその前にきちんと算数ができるように努力しないとね。

絵美「そのプログラムを使うのはだめ?」

まずは、算数の力をつけてからじゃないとね。
でも、電卓プログラムのしくみを勉強することはいいことだ。
算数の勉強にもなるから。

絵美・くま「プログラムのしくみって、なんだか面白そう。早く教えて!」



4-1 変数

変数とは

ここでは、算数プログラムを例に
プログラムが起動するまでの
コンピュータ内部のしくみをみていこう。

コンピュータ（電卓）が足し算などの計算をするときは、
入力装置を使って、

·1つめの数字（最初の数字）
·2つめの数字

の2つの数字をコンピュータに正しい方法で
命令を伝えてあげるということが大切になる。
このとき、私たちがコンピュータに命令を伝える場合、
「**コンピュータを使う側（ユーザ）**」と「**コンピュータ側**」の動きがポイントとなるんだ。これは、88ページの復習になるけれど、次の図に注目してコンピュータへ伝える方法をしっかりみていこう。

足し算を行う時の命令の流れ

ユーザ（人）	コンピュータ
① プログラムを起動する	ー
ー	②「最初の数字を打ち込んでください」と案内画面を表示する
③ 1つめの数字を打ち込む	ー
ー	④「2つめの数字を打ち込んでください」と案内画面を表示する
⑤ 2つめの数字を打ち込む	ー
ー	⑥ 計算結果を表示する

上記のように、
ユーザ（人）が入力装置（キーボードやマウス）を使ってプログラムを起動することで、コンピュータへ**足し算プログラムを実行する**命令を伝えることができる。
この結果、プログラムの通りに処理が進められていくんだ。

つまり、コンピュータを使う人が、
プログラムに必要なデータを
入力装置（キーボード）から入力することで、
必要な**数値データ**が**プログラムとして書き込まれ**、
書き込まれた順に処理が進み、

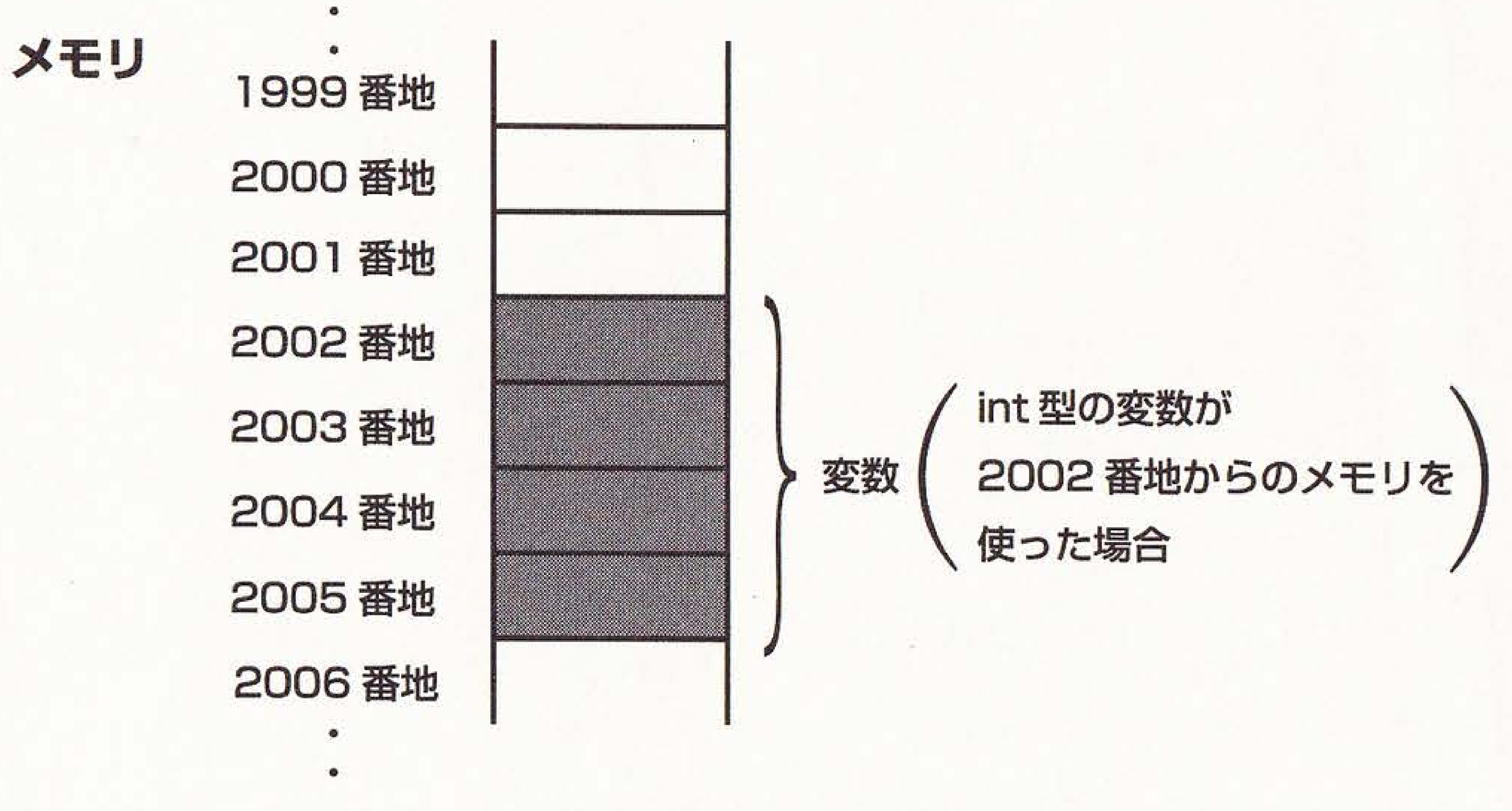
計算（足し算）ができるというわけだ。
このとき、足し算の「足す」という命令で使われた、
「1つめの数字」と「2つめの数字」というのは、
実際に足し算をする時のキー入力で**初めてわかる数**なんだ。

この時の打ち込んだ数値データは、**変数**という手段を使って、
記憶が行われる。

この数値データには、

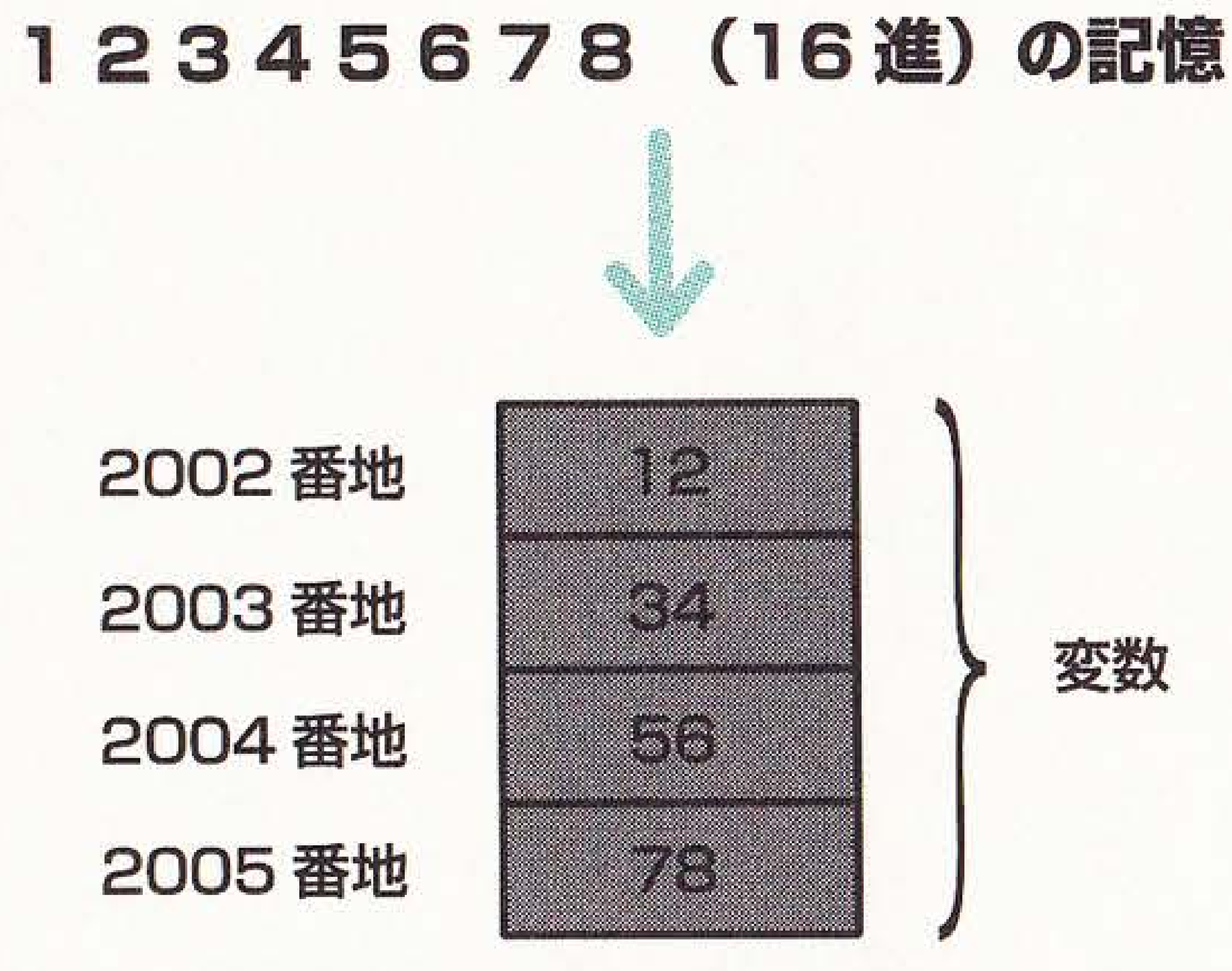
・**プログラムが動いた後**に決まる数値データ
・あるプログラムから**別のプログラムに伝える時**の数値データ
・プログラムが**計算途中**で覚えておきたい数値データ

などがあり、変数は、これらのデータを記憶しそれらを読み出したり、
書き換えたりするときに使うものなんだ。



変数とは

実際に、入力された数値データなどは
変数の機能を使って**メモリ**に一時的に記憶され、
処理が行われる。
そして、記憶されたデータは、必要なときに**特定のメモリから読み出す**ことができる。



変数は一時的に記憶される

メモリは、このキーから入力されたデータを**一時的に記憶するもの**なんだ。つまり、**メモリは一時的に保存ができる記憶装置**のことを指している。

人が計算するときには、複雑な計算になればなるほど、
メモに書いて計算するよね。
それと同じようにコンピュータも**メモのように覚える仕掛け**を持っておくんだ。

さっきの「1つめの数字」と「2つめの数字」も、メモリ（記憶装置）に変数をつくることで、一時的に保存がされている、ということになる。

このメモリは、1つひとつの記憶ができるメモを集めた、さしずめ大きなメモ帳といったところで、メモリ（メモ帳）は、これら1ページ、1ページのメモの集まりによってできているんだ。



● メモリはメモ帳のようなもの

それに、これらのメモの1つひとつは一度計算が終了しても、
今度は別の数値を入力して、異なる計算（足し算など）ができるようになっているんだ。
つまり、**読み書きが可能**で何回も書いたり消したりすることができる。
そして、次の計算をするときに数値の書き換えが必要になったら、
消しゴム（**電気スイッチのON/OFF**）を使って**元の数値を消し、新たな数値を記憶する**ことができるんだ。



メモリは読み書き可能

ポイント

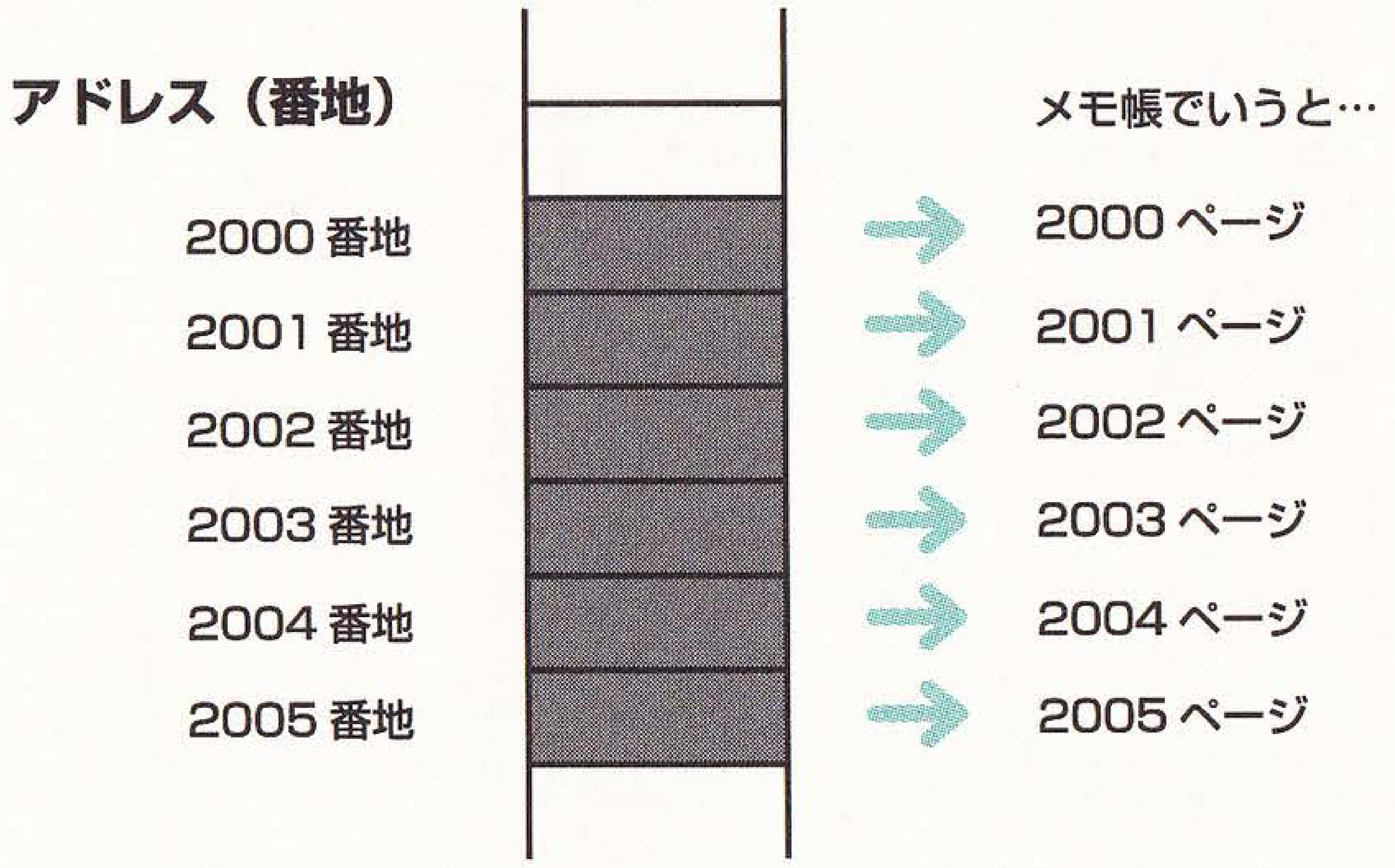
- メモリ上では、変数というしくみを使って数値データを一時的に記憶している
- メモリは、何度も読み書き可能な記憶装置である

変数とアドレス

変数は、メモリ（メモ帳）を利用してデータを記憶し、
処理を行う手段のことだったね。

さて、このメモリの中の記憶領域は、
たくさんのデータがあるので、**区画整理**される必要がある。
このとき、**アドレス**（**番地**）と呼ばれるものが、
記憶した場所を特定する働きをしているんだ。
アドレスは、メモ帳（メモリ）の中で何ページか（何番地か）を指し示す、**ページの番号**だと思ってもらえればいい。

プログラムの中では、いろいろなデータをメモし記憶していくけれど、
どのデータをどこに記憶したのかが分からなくならないように、
メモリ上にアドレス（メモ帳のページ番号）を示して、変数を使いやすくしているんだ。



● アドレスはどのページかを表わすもの



「どうしてそれが必要なの？」

プログラムの中で変数を使う時には、別のデータを読み出したりしてプログラムが動かなくならないようにしなければならない。
そのため、コンピュータが変数を使う時には、変数ごとにアドレスをきちんと決めて、メモリ上の位置を指し示すことになるんだ。



「どうやって、アドレスを指し示したらいいんだろう？」

パソコンのOSがメモリ上のアドレスを決めている。だから、私たちがプログラムを作る時には、変数の宣言をするだけでいいんだ。
実際には、1.変数の宣言をして、その変数を記憶する場所を決定し（**変数の宣言**，P137参照）、2.決まった場所にプログラムが動きはじめた時のメモの内容を記述する（**値の代入**，P141参照）ことがポイントだ。

このように、変数の記憶装置上の**アドレスと変数名を関係させる**ことで、プログラムが動くときに効率的にデータを処理することができる。

ポイント

- 変数を使う時には、変数ごとにアドレスを決めて、メモリ上の位置を示す
- 変数の記憶装置上のアドレスと変数名を関係させることで、効率的にデータを処理できる

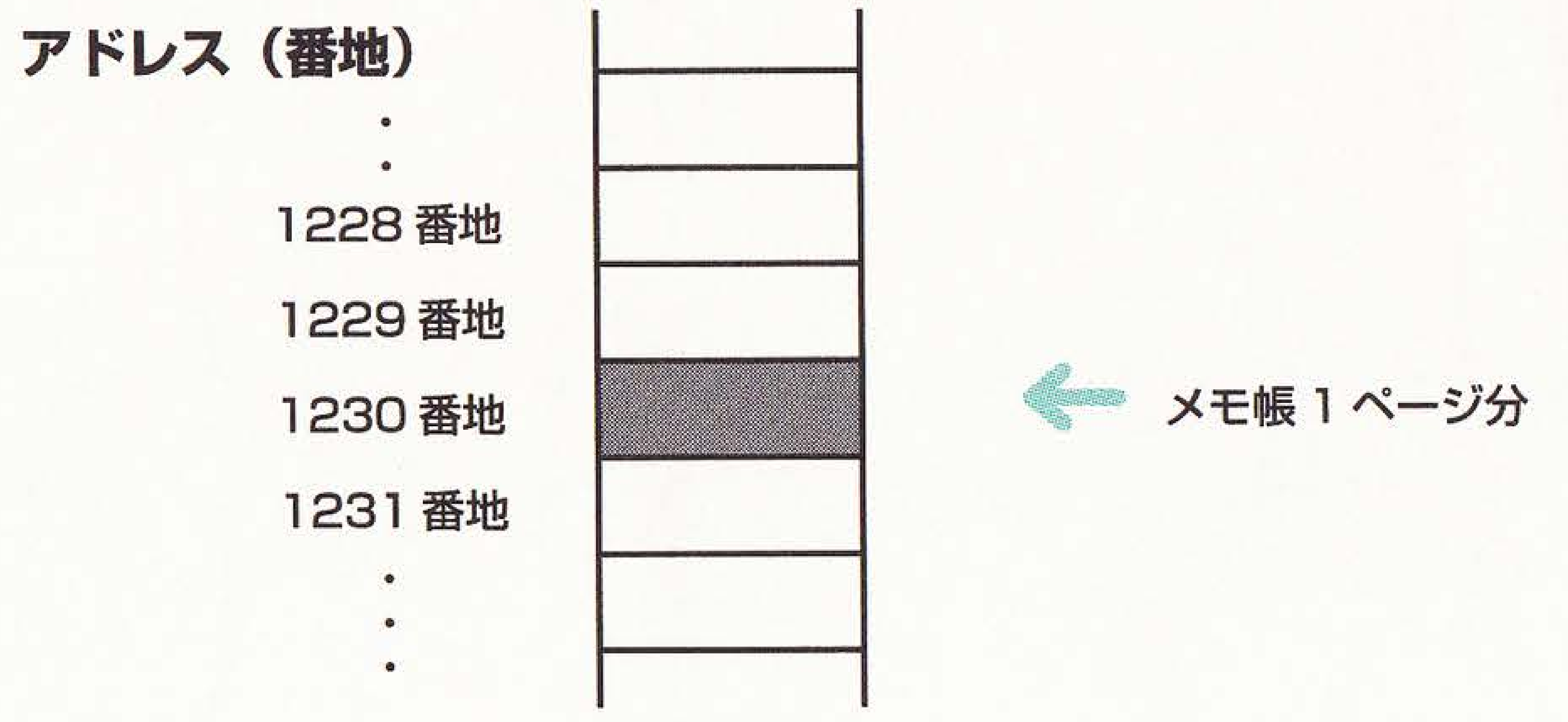
変数の型

変数は、その内容を記憶装置の決まったアドレスと関係させて記憶することにより、効率的にデータを処理しているんだったね（P123参照）。

そして、この変数を使うときには、一時的に記憶するデータの大きさを考える必要がある。このデータの大きさを**データ型**または**型**という。このデータの大きさによって、変数のありかを指し示す**アドレスの範囲が変わる**んだ。
たとえば、メモ帳の1ページに対して、型に対応するアドレスがどれだけのデータ量になるか、わかるかな?

実は、コンピュータの記憶装置に割り振られているアドレスは、1つのアドレスで**1バイト**の大きさの数値しか書けないんだ。
つまり、メモ帳1ページに書ける数値の大きさは、1バイトと考えて欲しい。

これは、メモリの**アドレス1番地分**の大きさに相当し、たとえば1230番地の1つを使うことになる。

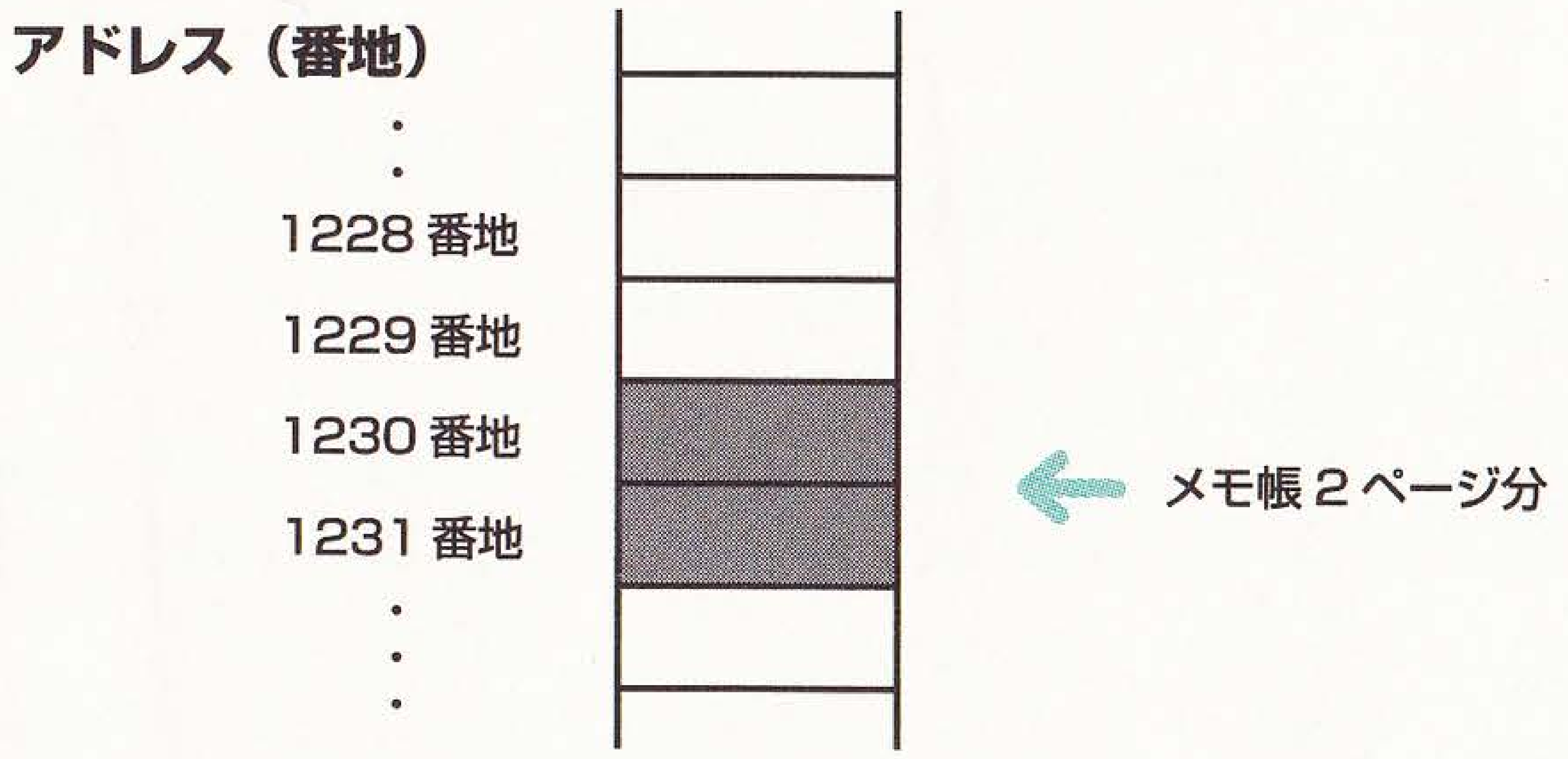


1つのアドレス=メモ帳1ページ分=1バイトの大きさ

もし、変数Xの大きさが2バイトだったら、メモ帳を何ページ使うことになるかな？

「1ページが1バイトだから、2ページは2バイトとすると…。」

そうだ、2バイトということは2ページ分のメモ帳を使うことになるね。つまり、アドレス2番地分の、1230番地と1231番地の2つを使うということになるんだ。



2つのアドレス=メモ帳2ページ分=2バイトの大きさ



データの大きさによってアドレスの範囲が変わる

また、変数aや変数bといった、複数の変数を使う場合は、それぞれを定義してメモ帳のページに当てはめるけれど、決まった大きさのデータをたくさん使いたい場合もある。
このような時には、変数の宣言をまとめてしまう方法もある。
これを「**配列**」というんだけど、配列で使うメモリは、配列として宣言して使うんだ（P184参照）。

では、データ型とそのデータのサイズについて説明していこう。

文字型	大きさ	数値の範囲
char	1バイト	−128〜127
unsigned char	1バイト	0〜255
整数型	**大きさ**	**数値の範囲**
int	4バイト	−2147483648〜2147483647
short int	2バイト	−32768〜32767
long int	4バイト	−2147483648〜2147483647
unsigned short int	2バイト	0〜65535
unsigned long int	4バイト	0〜4294967295
浮動小数点型	**大きさ**	**数値の範囲**
float	4バイト	3.4E−38〜3.4E+38
double	8バイト	1.7E−308〜1.7E+308
型なし		
void	返り値のない関数の宣言などに使う	

● データ型

次のようにデータ型には、様々な種類があるけれど、
変数のデータ型は、データの大きさが変わることによって、
プログラムで計算する時の計算結果に影響を及ぼすことがあるんだ。
簡単な算数プログラムに当てはめてその違いについて考えてみよう。

int型の宣言は・・・

・int saishonosuuji;
・int nibanmenosuuji;
・int kekka;

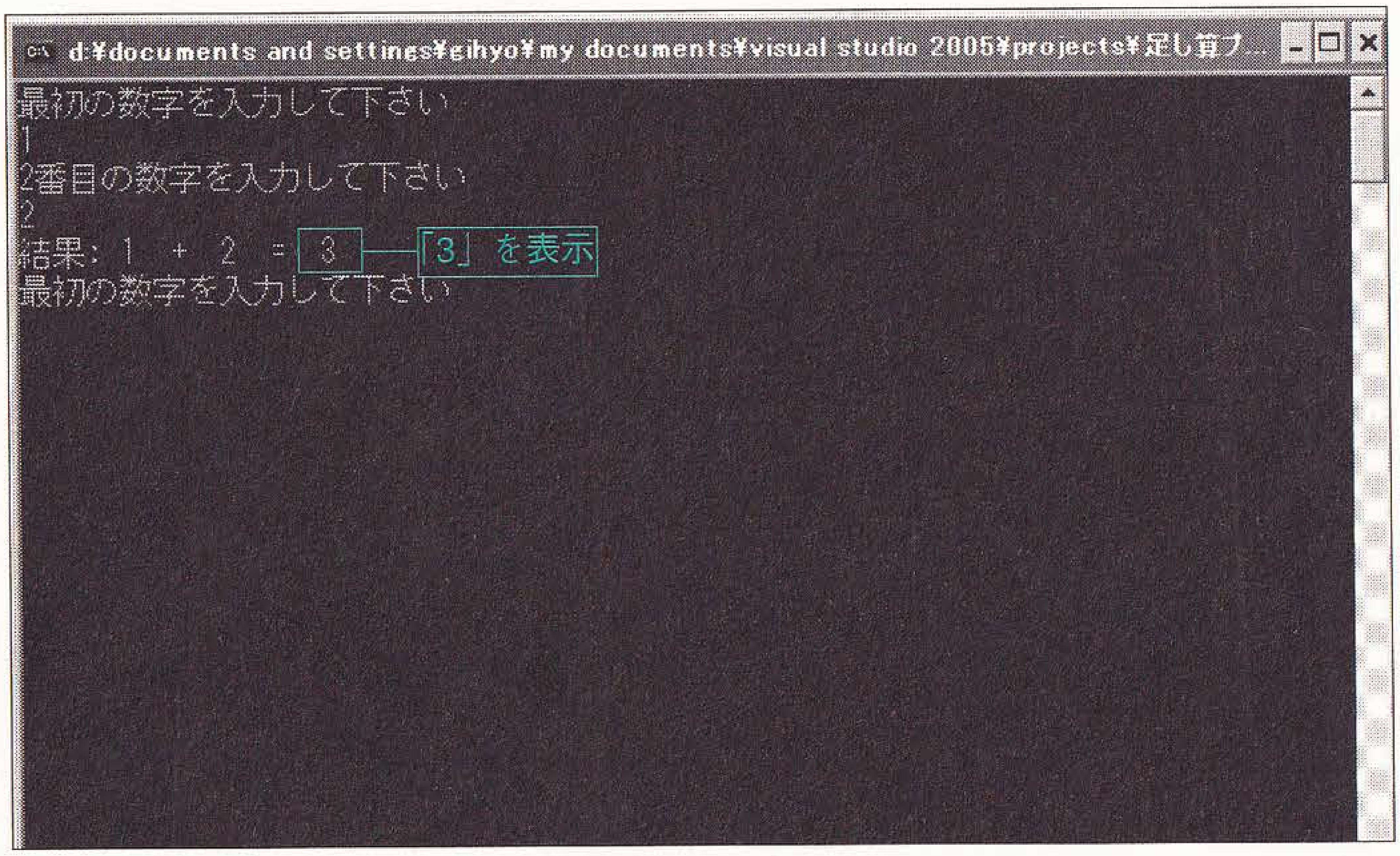
上記は整数型のint型で宣言しているけれど、int型のデータの大きさは、表の通り4バイトだ。
数値の範囲は「−2147483648〜2147483647」になる。
これをunsigned short int型に宣言し直してみることにしよう。

unsigned short int型に変更すると・・・

・unsigned short int saishonosuuji;
・unsigned short int nibanmenosuuji;
・unsigned short int kekka;

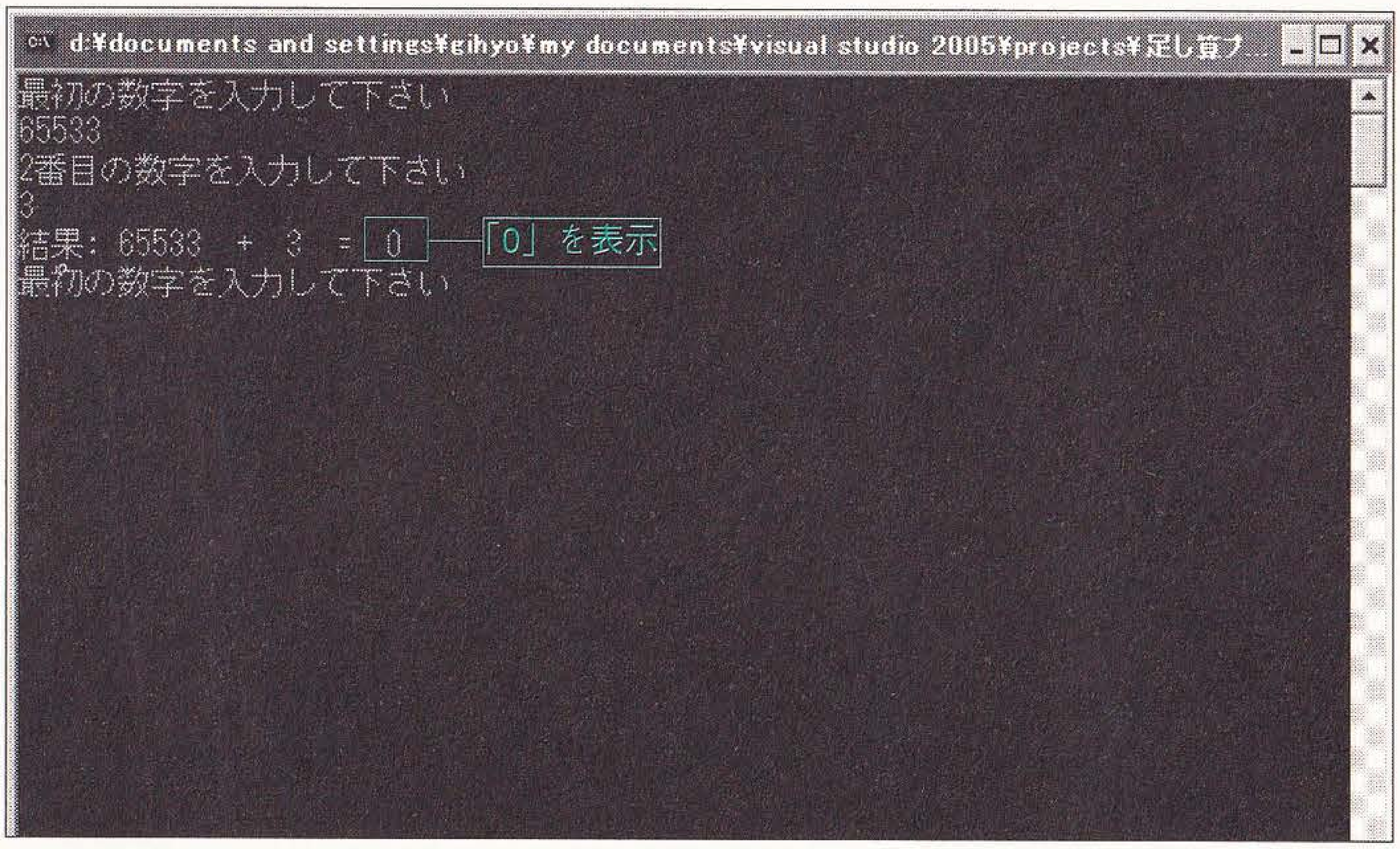
「int」から「unsigned short int」に変更することで、
データの大きさが「2バイト」に変わり、
数値の範囲は、「0～65535」になる。
データ型を変えることで、どのような計算結果になるのかみていこう。

・saishonosuujiを「1」にすると、
変数saishonosuujiに1が入る
・nibanmenosuujiを「2」にすると、
変数nibanmenosuujiに2が入る



● 変数「1」「2」を入れた時の計算結果（unsigned short int型）

コンピュータは、
saishonosuuji + nibanmenosuuji を計算して結果を表示するので
「1」+「2」の計算をして、「3」を表示する。
答えは3となり足し算はうまくできるね。
次に、入力する数字を大きくしたもので操作してみよう。
・saishonosuujiを「65533」
・nibanmenosuujiを「3」
とすると、答えが「0」と表示されるんだ。



● 変数に「65533」「3」を入れた時の計算結果（unsigned short int型）


「え〜! 65533+3=65536だから、0のはずがないじゃない。」

そう、0のはずがないね。



「データ型が変わったことで何が起きたの?」

問題が起こった理由をみていくことにしよう。

① saishonosuujiの「65533」は、unsigned short intの範囲の0～65535の間に入っている数値なので、問題なしだ。

② nibanmenosuujiの「3」も同じ範囲内なので、問題なし。

③ では、結果の「0」をみてみよう。

本来なら、変数kekkaには計算結果として「65536」が入るはずなんだけど、変数kekkaのデータ型を**unsigned short int**に変えたことでunsigned short int型の最大値65535を超えてしまった。これが、計算違いを生み出す原因となったんだ。こうなると、桁上がりして最上位一桁が欠けた数値、つまり0が結果として出されるんだ。

また、変数saishonosuujiや変数nibanmenosuujiは、unsigned short intで宣言しているので65535以上の数字を入れようとすると、その数字を65535で割った余りの数が表示されて正しい計算ができない。

もちろん、もとの変数定義（int型）のままでも、キー入力する数値の大きさがデータ型の上限（2147483647）を超えたり、
下限（－2147483648）を下回る場合には、同じように正しい計算ができないため注意が必要だ。



「これでは、電卓としては使えないね。」

つまり、変数の型宣言で宣言した、データ型の取り得る数値の範囲を超えるデータを扱おうとしたときには、その変数の数値は間違った値になってしまう、ということなんだ。

このように、データ型の**データの大きさは、プログラムの計算に影響を及ぼし**、計算結果を変えてしまうことがある。
だから、それぞれのデータ型の取り得る数値の範囲を知って、正しく使う必要があるというわけだ。



「データ型を決める時には、どんなことに注意すればいいんだろう…。」

実際に、お小遣いの計算を行う算数プログラムを例に、
データ型の決め方について考えてみよう。

まず、お小遣いの計算をするプログラムだったら、

・お金の「**単位**」
・お金の「**大きさ**」
・どの「**データ型**」にするか

を決める。

まず、お金の単位は「**円**」としよう。

次に扱うお金の大きさは、絵美のお小遣いや貯金の額の程度で決めることができる。銀行や証券会社ではないので、それほど多くはない、としよう。大体「**1000円〜1万円**」くらいだね。

最後にどのデータ型にするかは、プラスとマイナス分が必要になるかをまず考えよう。たとえば今月のお小遣いの残りが50円しかないとして、100円するものが欲しいような場合に、お母さんに来月分のお小遣いを前借りするよね。
そうした場合、今月分は使い切ってしまうと考えるため、お金は「**プラス分だけでなく、マイナス分も必要**」ということになる。

これらを考えて、

・単位・・・円
・使うお金の大きさ・・・小学生のお小遣いが1000円前後ということを考えて、数万円くらいは計算できる範囲とする
・データ型・・・前借りする場合もあるので、プラスとマイナスは使えたほうがよい

ということがわかるね。
これらの情報から、使えそうなデータの型を選んでいくよ。

扱うのは数値だから、
・文字型
・型なし
は、該当しない。

円の単位で計算できればいいので、
・浮動小数点型
は、該当しないね。

これら①、②を除くと残されたのは
・**整数型**
だね。

この整数型の中からプラスとマイナスの計算ができるものを考える。

①int（long intも同様：4バイト）で扱えるのは、「－2147483648」～「2147483647」なので、最大＋－21億円まで計算できるね。絵美のお小遣いには大きすぎて該当しない。

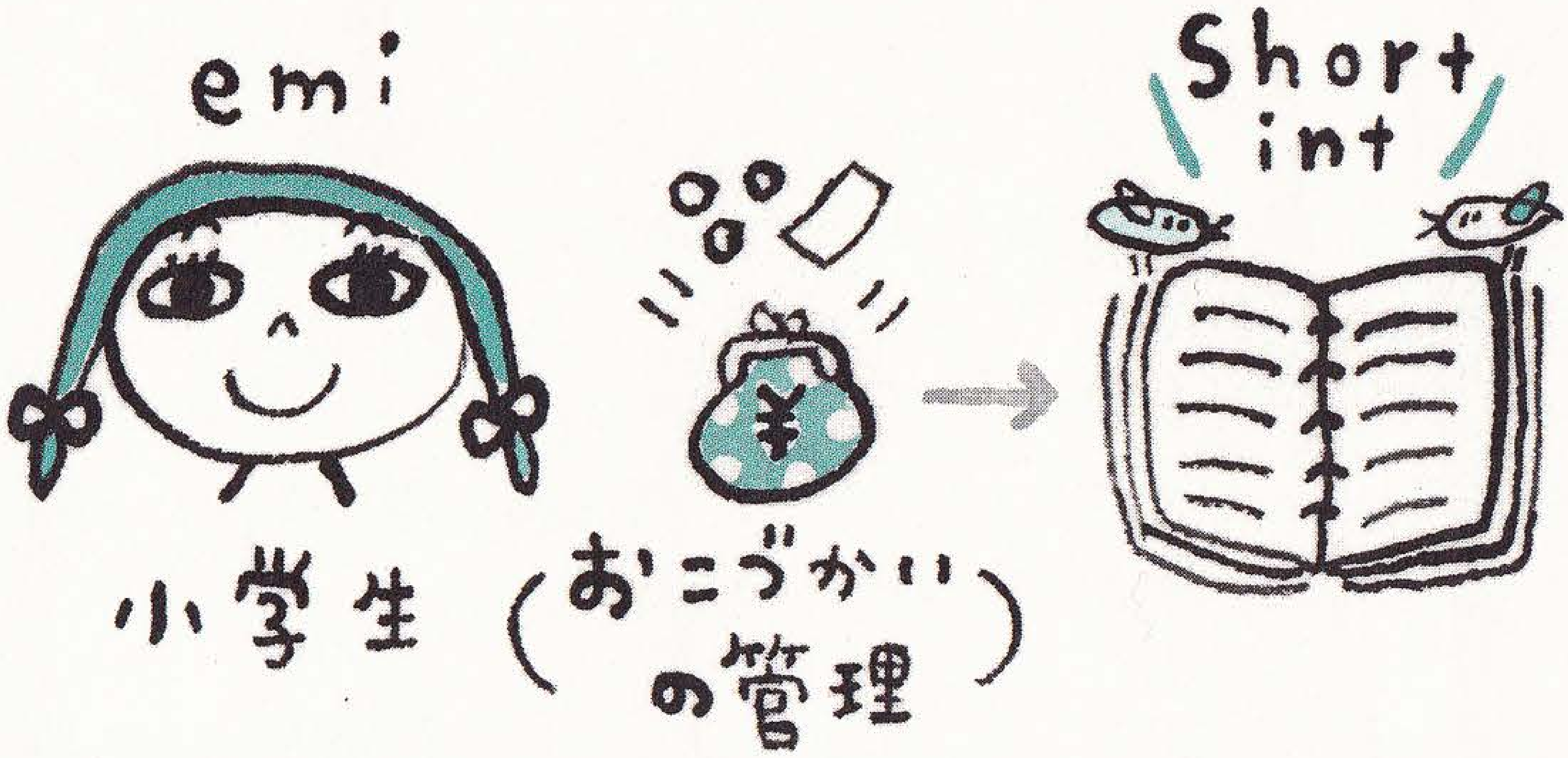
②short int（2バイト）で扱えるのは、「－32768」～「32767」で、32767円が上限になるね。絵美のお小遣いなら、short intでよさそう。

③unsigned short int（2バイト）で扱えるのは、「0」～「65535」で、マイナスが使えないから該当しないね。

④unsigned long int（4バイト）で扱えるのは、「0」～「4294967295」なので最大42億円まで計算できるけれど、数字が大きすぎて該当しない。

以上のことから、絵美のお小遣いの計算をするために適切なデータ型は、②の「**short int**」ということがわかる。

このようにデータの型は、そのプログラムを使う人の**目的**、**用途**を考えて決めることができる。お小遣いプログラムは、絵美がお金を管理する目的でつくる。用途は、プログラムの使い道のことで、お小遣いの計算をするということになる。つまりプログラムで使う変数のデータ型は、日々使うお金の状況に応じて必要な**データのサイズ**や**属性**（プラス、マイナス、小数点は必要であるか、そうでないか）を考慮して決めていくんだ。



● データ型を決める時は目的、用途を明らかに･･･

ポイント

- データの大きさをデータ型という
- データの大きさによって、変数のありかを示すアドレスの範囲が変わる
- データの型は、プログラムを使う人の目的、用途を考慮して決める

変数の名前（識別子）

これまで、変数やデータ型などを学んできたけれど、
変数には名前があるんだ。

変数の名前の例として、次のようなものがある。

```
short int saishonosuuji;
short int nibanmenosuuji;
short int kekka;
```

変数名

「saishonosuuji」「nibanmenosuuji」「kekka」という名前が出てきたね。
これらは、変数の名前として扱われているんだけれど、
変数の名前の付け方を少し考えてみることにしよう。

変数は、プログラムの中で使用するものなので、
プログラムを見る人にとって、誰にでもわかりやすい名前になっていたほうがいいよね。
ただし、ここで問題なのは、プログラムの中では、英字や数字しか使えないということだ。

プログラムにある**saishonosuuji**という変数名が、**最初の数字**と書ければ、よりわかりやすいんだろうけれど、変数の名前は英字や数字しか使えないというきまりがある。

ここでは、変数名の基本的なきまりを覚えておこう。

・英字、数字、アンダースコアのいずれかを使う
・予約語（あらかじめC言語が予約しているキーワード）は使えない（例：returnなど）
・数字ではじめることはできない
・大文字と小文字は別の変数として区別する
・あまり長い変数名は避ける。長くても30文字程度に収める

また、変数名から変数の意味や使われ方が間違わずにわかるような名前を付けるということが大切なんだ。

例えば「**kekka**」から、何が想像できるかな?



「結果でしょ、だから、計算結果のこと。他の意味は特になさそう。」

そうだね。変数の命名がうまくいっているということだ。
変数の名前を付けるときには、いろんな解釈ができる言葉は使わないほうがいいね。プログラムを見る人によって意味が変わってしまうとプログラムが理解できなくなるし、この変数は一体何なのかをよく調べないとならないからね。このように変数の名前を付ける時には、プログラムを扱う人のことを考えることが大切なんだ。

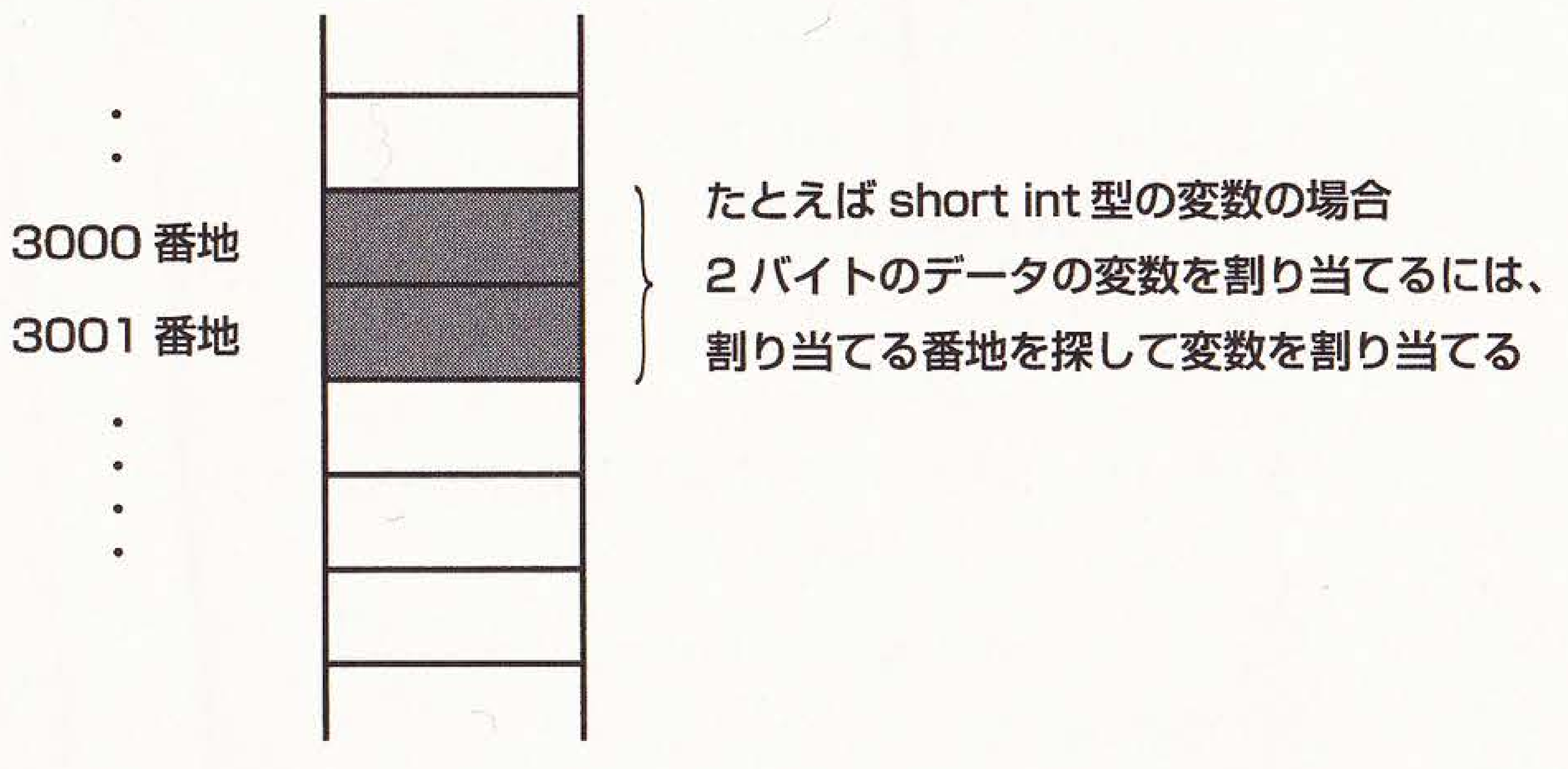
ポイント

- 変数名を付けるときには、基本的なきまりを守ることが大切である
- 変数名は、目的や用途がすぐに想像できるような名前にする

変数の宣言

変数を利用するには、**変数の宣言**と**値の代入**が必要になるんだ。
まず変数の宣言は、**変数の型**や**変数の名前**をつけることをいう。

これは、プログラムが計算をする時に使う変数を
コンピュータがそれぞれの変数ごとに、**記憶装置の中の決まった場所にその変数を割り当てるように指示する**ことを意味する。
このようにすることで、割り当てられた場所に自分の使う変数のアドレスを決めることができるわけなんだ。



変数の割り当て

つまり、変数の宣言は、メモ帳（メモリ）（P120参照）のどのページに変数を割り当てるかを決めるということなんだ。コンピュータはメモ帳のそれぞれのページをアドレスで管理しているので、そのページがわかれば、該当するデータをすぐに見つけ出せるんだ。

実際に、算数プログラムの「変数の宣言」がされている部分をみてみよう。

リスト4-1　算数プログラム

```
#include <stdio.h>          //標準関数を使用するための宣言
void main () {
            short int saishonosuuji;
            //変数saishonosuujiの宣言
            short int nibanmenosuuji;
            //変数nibanmenosuujiの宣言
            short int kekka;
            //変数kekkaの宣言
while (1) {
        printf ("最初の数字を入力して下さい ¥n") ;
        //最初のコメント表示
        scanf ("%d",&saishonosuuji) ;
        //最初の変数を入力
        printf ("2番目の数字を入力して下さい ¥n") ;
        //2つめのコメント表示
        scanf ("%d",&nibanmenosuuji) ;
        //2つめの変数を入力
        kekka = saishonosuuji + nibanmenosuuji;
        //計算し、結果を変数kekkaへ入れる
        printf ("結果:") ;
                    //結果の表示
```

変数の宣言

値の代入

```
13          printf (" %d ", saishonosuuji) ;
14          printf (" + ") ;
15          printf (" %d ", nibanmenosuuji) ;
16          printf (" = ") ;
17          printf (" %d ", kekka) ;
18          printf ("¥n") ;
            //結果の表示を終了し、改行

19                  }
20          }
```

03～05行目にあるコメント文をみてみよう。

```
03          short int saishonosuuji;
            //変数saishonosuujiの宣言
04          short int nibanmenosuuji;
            //変数nibanmenosuujiの宣言
05          short int kekka;          //変数kekkaの宣言
```

つまり、このプログラムは、03、04、05行目で3つの変数を使うための宣言をしているんだ。
このようにすることで、コンピュータは**記憶装置の決まった場所に自分の使う変数のアドレスを決める**ことができ、メモ帳（メモリ）の、あるページに変数を使う準備が整う。

このプログラムで使う変数は、すべてshort int型で宣言しているので、それぞれ2バイトとして考えると、合計6バイト分の記憶領域が確保される、というわけなんだ。
変数の宣言をする時は、

1. 必要な型を決める
2. 変数に名前をつける

の順番で行う。書き方は、使う変数名の前にその変数のデータ型（データ型については、P124参照）を書くだけなんだ。

short int saishonosuuji〔**データ型**〕〔**変数名**〕

こうすることでコンピュータはプログラムを実行したときに、このプログラムのための記憶領域を「saishonosuuji」という変数のためだけに使えるように設定ができ、この**記憶領域に書かれているデータの内容を扱うことができる**わけなんだ。

ポイント

- 変数の宣言は、プログラムの中で使用する変数を使うための準備をする
- プログラム上では、使用する変数の記述よりも手前で、変数の宣言する
- 変数の宣言をすれば、宣言した変数名は変数として認識され、その記憶領域に書かれているデータの内容を扱うことができる

値の代入

変数の宣言をすることで、その変数に割り当てられた記憶場所に特定の値を記憶させることができる。このことを「**値の代入**」という。
算数プログラム（P138参照，リスト4-1）をもとに、値の代入をみていこう。

```
08      scanf ("%d",&saishonosuuji);
        //最初の変数を入力
```

ここでは、コメントに「//最初の変数を入力」とあるね。この08行を実行することで、変数「saishonosuuji」にキーボードで打ち込んだキーの数値データが入ることになるんだ。
つまり、scanf（"%d",&saishonosuuji）;と書くことで、
変数名で示すsaishonosuujiにキー入力したshort int型のデータが入る。ちなみにscanf（ ）は標準入力であるキーボードからデータを取り込む関数を示している。
次に下のコードをみてみよう。

```
10      scanf ("%d",&nibanmenosuuji);
        //2つめの変数を入力
```

これもsaishonosuujiと同じように、変数nibanmenosuujiにキーボードから打ち込んだ数値データがshort int型のデータとして入ることを示しているんだ。

```
11      kekka = saishonosuuji + nibanmenosuuji;
        //計算し、結果を変数kekkaへ入れる
```

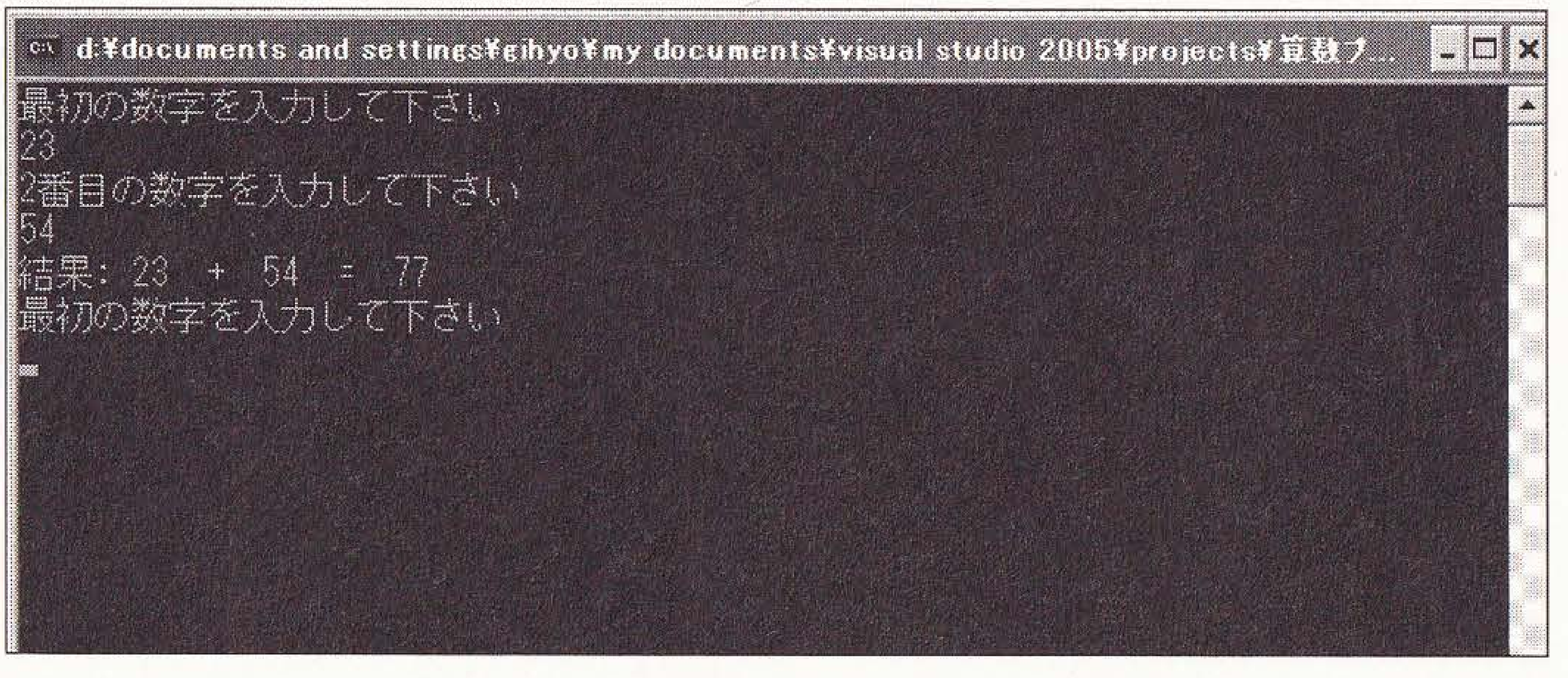
11行目では、これまでに決定した変数の数値を計算している。
つまり計算結果は、short intで宣言した変数kekkaに入ることになる。
また、プログラムの「=」は、算数の「=」と違って**「=」の右側（右辺）の数や計算結果を左（左辺）の変数に入れる**という命令になるんだ。

このように変数を使う時には、必ず「値の代入」を行っているんだ。
続けて、コードをみていこう。

```
13        printf(" %d ", saishonosuuji);
15        printf(" %d ", nibanmenosuuji);
17        printf(" %d ", kekka);
```

これらのprintfは、scanf（ ）関数で取り込んだデータを標準出力であるディスプレイに表示（出力）する働きをしている。

それぞれsaishonosuujiは、1番目に入れるデータを、nibanmenosuujiは2番目に入れるデータを表示するんだ。また、変数kekkaには、1番目と2番目の足し算の結果を表示することを表わしている。



取り込む数値データと計算結果の出力

この時プログラムで扱う変数は3種類になり、
変数saishonosuujiと変数nibanmenosuujiは、入力された数値そのものを記憶して、変数kekkaは、2つの変数を足した結果を記憶するために使われるということがわかるね。

このようにコンピュータでは、変数（記憶装置）に値が代入され、プログラムに書いてある順番に処理がされて、結果を表示させる働きをしているんだ。

ポイント

- 変数に割り当てられた記憶場所に値を記憶させることを、値の代入という
- 変数を使うときには、変数に値を代入し、プログラムの順番に処理を行うことで結果を出すことができる

4-2 演算子

演算子とは

これまで算数プログラムを使って値の代入や計算結果の表示について学んできた（P141参照）。
このときコンピュータでは、決まった数値や変数を使って計算を行っているんだ。
これらの計算で用いる記号のことを**演算子**という。



「たとえば、演算子の記号にはどういうものがあるの?」

・足し算の「＋」（プラス）
・引き算の「－」（マイナス）
などがあるんだ。



「それって、足し算をするのか、引き算をするのかを決める記号のこと?」

そう、計算の種類を決める記号のことだね。

プログラムを書くためのC言語では、たくさんの演算子が準備されていて、いろいろな計算ができるようになっているんだ。
それでは、プログラムの中の演算子を確認してみよう（P138参照）。

```
kekka = saishonosuuji + nibanmenosuuji;
 //計算し、結果を変数kekkaへ入れる
```

「算数プログラム」に含まれている演算子は、
＋演算子と呼ばれるもので、この1行だけだ。



「たった1行で計算ができるの!?」

うん、「＋」の演算子がさえあれば
足し算ができるんだ。

それに、もうひとつの演算子があるのはわかるかな?



「えっ、もしかして、＋以外というと、「＝」ですか?」

そうだ。

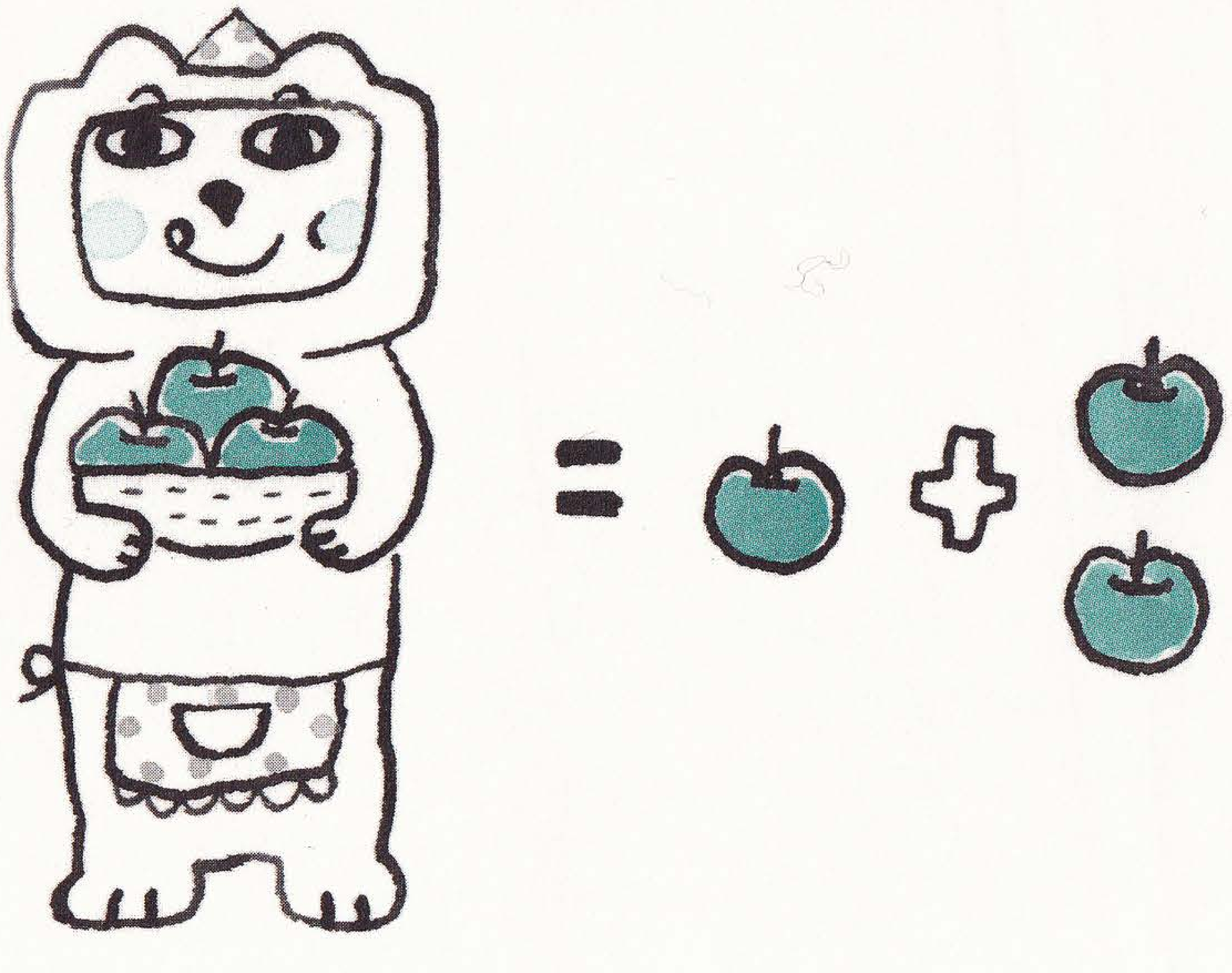
この「＝」は、算数と同じく計算結果を出すときに使うものだね。
算数の場合は、「＝」を計算式の右側に書いて、続けて右側に計算結果を書くよね。
でもプログラムの場合は、「＝」の左側（左辺）の変数に「＝」の右側（右辺）の演算結果を代入（コピー）することを意味し、この**「＝」を代入演算子**と呼ぶんだ。

つまり、

```
kekka = saishonosuuji + nibanmenosuuji;
```

の計算の中で、「＋」は、手前の数字（変数saishonosuuji）と後の数字（変数nibanmenosuuji）の2つの変数を足し算（加算）するということを示している。

そして「＝」は、右側のsaishonosuuji + nibanmenosuujiの計算をして、その計算結果を「＝」の左側の変数kekkaに代入する（コピーする）ということを示しているんだ。



● 代入演算子

また、変数a＝変数bのように書く場合もあるんだけど、このときは「＝」の右の「変数b」の内容を左の「変数a」に代入するということを意味しているんだ。


「じゃあ、「×」や「÷」はどういうきまりがあるの?」

掛け算、割り算は足し算、引き算より優先され、最初に出てきたものから順番に計算するんだ。

たとえば、
3×4＋6÷2－2×2＋8÷2÷2

の計算をするなら、①～⑥の順に計算をして結果を出すことになる。

1.　3×4＝①
2.　6÷2＝②
3.　2×2＝③
4.　8÷2÷2＝④
5.　①＋②＝⑤
6.　⑤－③＝⑥
7.　⑥＋④


「これは、算数ですでに習ったよ。」

そうだね、算数のルールと一緒だね。
簡単なことだけれど、「×」「÷」は、これらの順序で計算が行われていくんだ。ちなみに、コンピュータで扱う演算子の場合には、「×」は「＊」、「÷」は「／」で表すよ。



「計算順序を守れば、計算結果が出されることになるのね。でも、一体、計算途中の結果はどうしているの?」

するどい質問がきた。

計算式
3×4＋6÷2－2×2＋8÷2÷2

の計算では、

1.　　3×4＝①12
2.　　6÷2＝②3
3.　　2×2＝③4
4.　　8÷2÷2＝④2
5.　　①＋②＝⑤15
6.　　⑤－③＝⑥11
7.　　⑥＋④＝13

という順序で計算結果を出すという説明をしたけれど、この計算の場合には、変数として用意したメモ（メモリ）以外に丸数字で示した①～⑥の6つのメモ（メモリ）を用意する必要がある。

「“6つのメモ” って何？」

コンピュータがプログラムを動かす時には、プログラムを実行する時に必要になるメモ用として“**ワークメモリ**”というメモリを準備しているんだ。
つまり、①〜⑥に当てはまる計算途中の数字は、プログラム実行用のワークメモリに記憶しながら計算を進めていることになるんだね。

ポイント

- コンピュータは、計算をするために作られているので、この計算順序を守ることで、正しい結果を出すことができる
- 演算子は、四則演算のきまりを守りつつ、計算順序に従って計算をしていく

いろいろな演算子

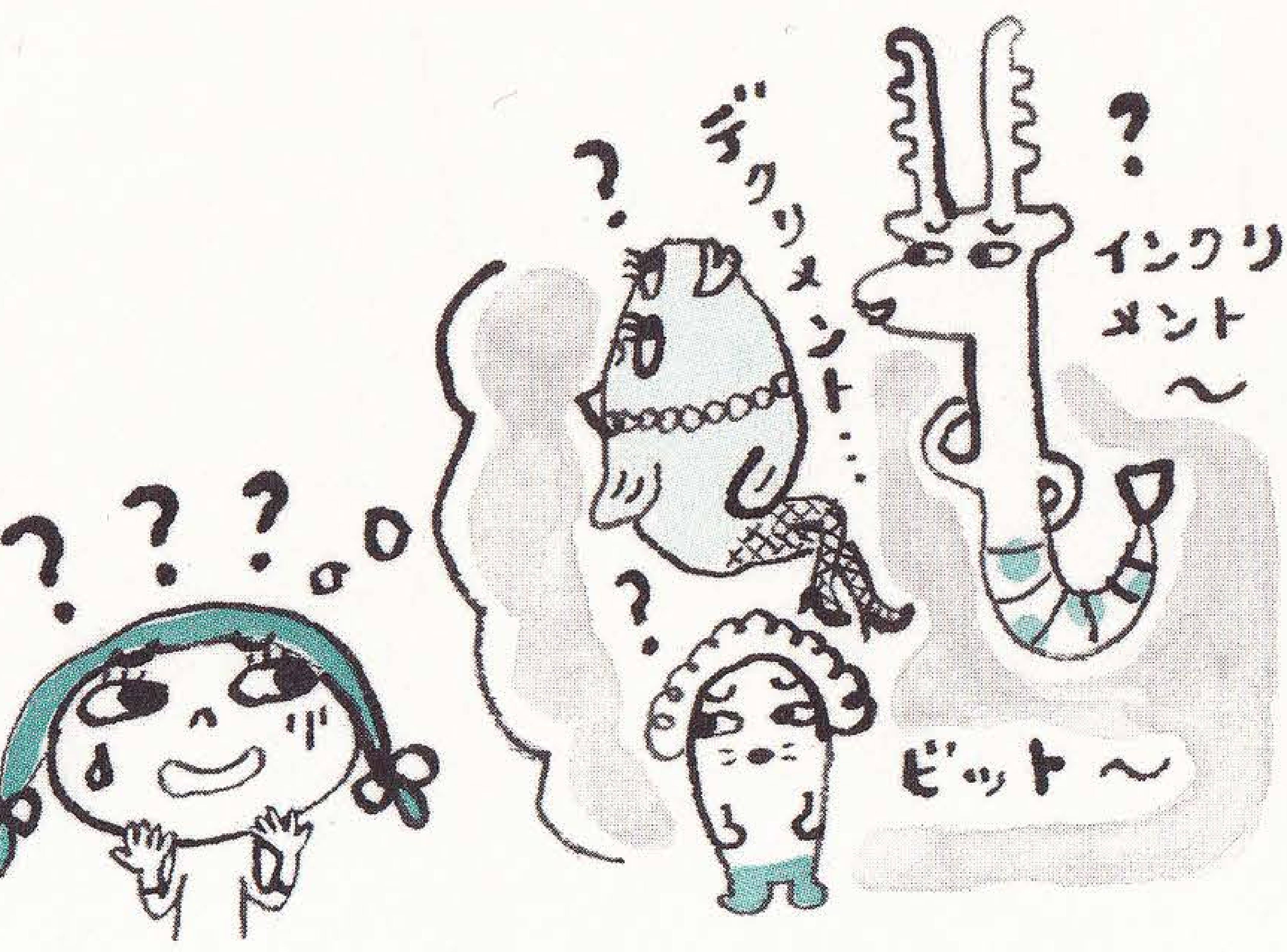
演算子の計算で使われていた記号（P138参照）には、
「＋」「－」「×」「÷」など基本的な演算子があったね。
この他にも様々な種類の記号があるので演算子についてまとめてみていこう。

この演算子には、複合代入演算子、インクリメント演算子、デクリメント演算子、ビット演算子、関係演算子、論理演算子などが準備されている。

「なんだか、聞いたことのない演算子ばっかり。」

「ずいぶん、たくさん用意されていますね。」

まあ、コンピュータは計算を行うための道具なので、当然といえば当然なんだけど、それぞれの演算子の機能をよく理解してから使わないときちんとした結果が得られないので、注意が必要だ。



・代入演算子

代入演算子は、演算子の右辺または、右辺と左辺の数値による演算の結果を左側の変数に代入する命令だ。計算するときは、必ず**代入演算子を使って、結果を変数に取り込む**んだよ。

（例）kotae＝saishonosuuji + nibanmenosuuji ;

例の場合は、変数kotaeに変数saishonosuujiと変数nibanmenosuujiを足した結果を取り込むことを示している。

代入演算子

演算記号	記号の意味と使い方
＝	（右辺）を（左辺）に代入する。右辺が演算記号を含む場合には、演算結果を左辺に代入する
＋＝	（右辺）＋（左辺）の計算結果を（左辺）に代入する
−＝	（右辺）−（左辺）の計算結果を（左辺）に代入する
＊＝	（右辺）×（左辺）の計算結果を（左辺）に代入する
／＝	（右辺）÷（左辺）の計算結果を（左辺）に代入する
%＝	（右辺）÷（左辺）の計算結果の余りを（左辺）に代入する
&＝	（右辺）と（左辺）の論理積を（左辺）に代入する
｜＝	（右辺）と（左辺）の論理和を（左辺）に代入する
^＝	（右辺）と（左辺）の排他的論理和を（左辺）に代入する
＝~	（右辺）の1の補数を（左辺）に代入する
<<＝	（右辺）で指定したビットだけ（左辺）を左シフトした数を（左辺）に代入する
>>＝	（右辺）で指定したビットだけ（左辺）を右シフトした数を（左辺）に代入する

・インクリメント演算子
インクリメント演算子は、**変数を＋1する**命令のことをいい、
変数を数える場合に用いる。

（例）hensuu++；（hensuu = hensuu +1 ;）

これは、変数hensuuの値に1を足して、変数hensuuに入れること
を示している。

インクリメント演算子

演算記号	記号の意味と使い方
++（変数）	（変数）に+1する
（変数）++	（変数）に+1する

・デクリメント演算子
デクリメント演算子は、**変数を－1する**命令のことをいい、
変数を数える場合に用いる。

（例）hensuu−−;（hensuu = hensuu −1 ;）

これは、変数hensuuの値から1引いて、変数hensuuに入れること
を示している。

デクリメント演算子

演算記号	記号の意味と使い方
−−（変数）	（変数）に−1する
（変数）−−	（変数）に−1する

・関係演算子

関係演算子は、「**成立すれば真**」「**成立しなければ偽**」といった処理を意味している。if文などの条件文として使い、**真になれば、直後の {} 内の処理を行う。偽になればその処理は行わない**ことを意味しているんだ。
変数の数値によって分岐の判断をする時に用いる。

（例）
```
if ( ookii > chiisai ) {
    ookii = chiisai ;
}
```

この例では、if文に続く（　）の中が条件式で、変数ookiiが変数chii-saiよりも大きければ、条件が成立して“真”になり、{ から } までの処理を行うことになる。
条件が成立しなければ、結果は“偽”で、{ から } までの処理は行わずに次の処理に進む。

関係演算子

演算記号	記号の意味と使い方
<	（右辺）より（左辺）が小さい時に真
>	（右辺）より（左辺）が大きい時に真
<=	（右辺）より（左辺）が小さいか等しい時に真
>=	（右辺）より（左辺）が大きいか等しい時に真
==	（右辺）と（左辺）が等しい時に真
!=	（右辺）と（左辺）が等しくない時に真

・論理演算子

論理演算子は、複数の関係演算子を組み合わせて**真と偽を判断する**命令のことをいうんだ。
複数の関係演算子の結果を使って、処理を変更するときに使う。

（例）
```
if ( ( ookii == chiisai ) || ( ookii < chiisai ) ) {
    ookii++ ;
  }
```

この場合、複数の関係演算子を使って、条件を作る時に使用する。最初の条件(ookii == chiisai)と、次の条件 (ookii < chiisai)が両方とも成立したら、変数ookiiを＋１する。
またこの例で示した条件式は、

```
if ( ookii <= chiisai ) {
```

と書いた時と、同じ処理になることを示している。

論理演算子

演算記号	記号の意味と使い方
!	条件式と一致しない時に真
&&	（右辺）と（左辺）両方の条件式が真の時に真
||	（右辺）と（左辺）どちらかの条件式が真の時に真

・ビット演算子

ビット演算子は、数値データをビット単位で操作する場合の、**演算を指示する**命令のことをいうんだ。
ビット単位でのデータの確認が必要な時などに使う。

（例）hensuu = hensuu ^ hensuu ;

この例は、すこし難しいよ。左辺の変数hensuuには、右辺の変数hensuuと変数hensuuの**排他的論理和**（2つの数字のビット同士を比較し、同じだったら0、違ったら1にする）の計算結果が入る。同じ変数の排他的論理和を計算すると、答えは0になる。よって、書き換えると、

```
hensuu = 0 ;
```

と同じ意味になるんだ。

ビット演算子

演算記号	記号の意味と使い方
&	（右辺）と（左辺）の論理積
|	（右辺）と（左辺）の論理和
^	（右辺）と（左辺）の排他的論理和
~	1の補数、つまり、ビット反転
<<	指定したビットだけ左シフト（つまり×2n）
>>	指定したビットだけ右シフト（つまり÷2n）

このように、演算子の中には聞き覚えのないものがたくさん準備されているけれど、コンピュータに計算をさせる時に作るプログラムの演算子は、算数や数学で使う演算子と違うことをよく理解しておく必要があり、演算子の働きや機能をよく知っておくことが大切だ。

ポイント

演算子は、その働きや機能をよく理解してから効率的に使うことが大切である

4-3 制御文

if文

プログラムを使い、コンピュータに仕事をしてもらう時に
データの内容によっては、別の処理をしたいことがよくあるよね。

たとえばキーボードで、次のような条件付きの処理があるとしよう。

・はい（Y）と打ち込んだ場合には、
計算を始める

・いいえ（N）と打ち込んだ場合には、
計算をしないで入力画面に戻る



「これらの処理は、どうやって変えることができるんですか?」

プログラムには、変数の値と数値を比較して、
処理を変えることができるしくみがあるんだよ。

このしくみのことを**条件分岐**といい、
条件分岐を行うプログラムのことを**条件文**というんだ。

たとえば、
「変数名をhensuuとして、変数hensuuが「50」と等しかったら、
その変数hensuuに「0」を入れる。それ以外なら何もしない。」
というプログラムを考えてみよう。

```
01    if (hensuu == 50) {
02        hensuu = 0;
03    }
```

01行目にある「if」は、条件分岐を代表するif文だ。
「**==**」は演算子の仲間で「**関係演算子**」と呼ばれ、( )の中の式全体は、
条件式と呼ばれるものなんだ（P153参照）。



「いふぶんって何?」

このif（いふ）は、
「もしも〜だったら･･･する」っていうことを表していて、
〜だったらの〜に相当する部分が、ifに続くかっこ（ ）の中に書いてある。

（ ） の中のhensuu == 50というのは、
「hensuuが50と等しかったら」という条件を示すもので、
これは計算に使う演算子とは少し違って、演算結果が正しいか、正しくないかで示される。

このとき演算結果が正しければ、続く { } の中の処理を実行し、
演算結果が正しくなければ、{ } の中の処理は実行せずに、その次の行を実行することができるんだ。

つまり、01行目のifの後にあるカッコ（ ）の中で、hensuuの値が「50」と等しければ（このことを**条件式が成立**という）、変数hensuuを0にするということを表わしているね。



「もし、hensuuが50ではなかったらどうするの?」

変数hensuuが50ではなかった場合には、03行目の } の次の処理を行うことになるんだけれど、このプログラムではその先は書いていないので「何もしない」ということになる。
これがこの関係演算子の働きだ。

ポイント

- 変数の値と数値を比較して処理を変えるしくみのことを条件分岐という
- 条件分岐を行うプログラムのことを条件文という

if～else文

制御文のif文は理解できたかな?
次の制御文は**else**というもので、**if**とセットになっているものなんだ。

else文のあるプログラム

```
01  if (hensuu == 50) {
02  hensuu=0;             //①条件が成立した時に行う
03  }
04  else{
05  hensuu=hensuu+1;      //②条件が成立しない時に行う
06  }
07
```

このプログラムでは、変数hensuuの値が50でifの条件式が成立した時には、その後の{①}を処理するため、変数hensuuの値は0になり、07行目に進む。一方、ifの条件式が成立しない時には、elseの後の{②}を処理するため、変数hensuuの値は+1されるんだ。
つまり、この2つの処理は、どちらか一方だけ行うというわけなんだ。
条件式が成立した時と条件式が成立しなかった時とで、やりたいことが異なる場合には、このようなしくみがあると便利なんだ。

これに対して、else文のないプログラムは、次のようになっている。

else文のないプログラム

```
01 if (hensuu == 50) {
02         hensuu = 0;
03 }
04         hensuu=hensuu+1;
```

変数hensuuの値が50で条件式が成立した場合は、02行目で変数hensuuに0を代入する。その後に04行目を行うので、変数hensuuは+1した値、つまり、1になるんだ。
よって、この一連のプログラムを実行した後、変数hensuuの値は1になる。

つまり、else文のあるプログラムでは、条件式が成立した場合とそうでない場合とで処理を変えることができるのに対し、else文のないプログラムでは、プログラムの順番に与えられた1つの処理がされていくことを示している。

ポイント

if～else文は、条件が成立した場合としない場合とで、プログラムが進むべき順番を変えることができる便利な制御文である

while文

次に繰り返しの処理を行う条件文をみてみよう。

```
01    while (hensuu != 50) {
02        hensuu=hensuu+1;    //繰り返し処理を行う
03    }
```

これは、while文といって、やはり制御文だ。
while文の後の（ ）の中は、この条件式が成立するとその後の｛｝の中の処理を繰り返すことを意味している。

このプログラムでは、01行目の条件式は、
変数hensuuが50以外のときに成立するという意味を持つ。
そして、この条件が成立しなくなるまで02行目の処理を繰り返し行うんだ。
同じ処理を繰り返したい場合には便利な機能だね。

ポイント

while文は、条件式が成立した場合に、繰り返し処理を行う働きのある制御文である

4-4 関数

関数とは

今まで勉強してきたこのプログラムの本は、C言語を基に書いている。けれど、C言語はすべて関数から成り立っているんだ。
つまり、C言語は関数が集まったものといえる。

次のコードは、関数のひとつである**処理のかたまり**のことを指している。

```
void main () {
```

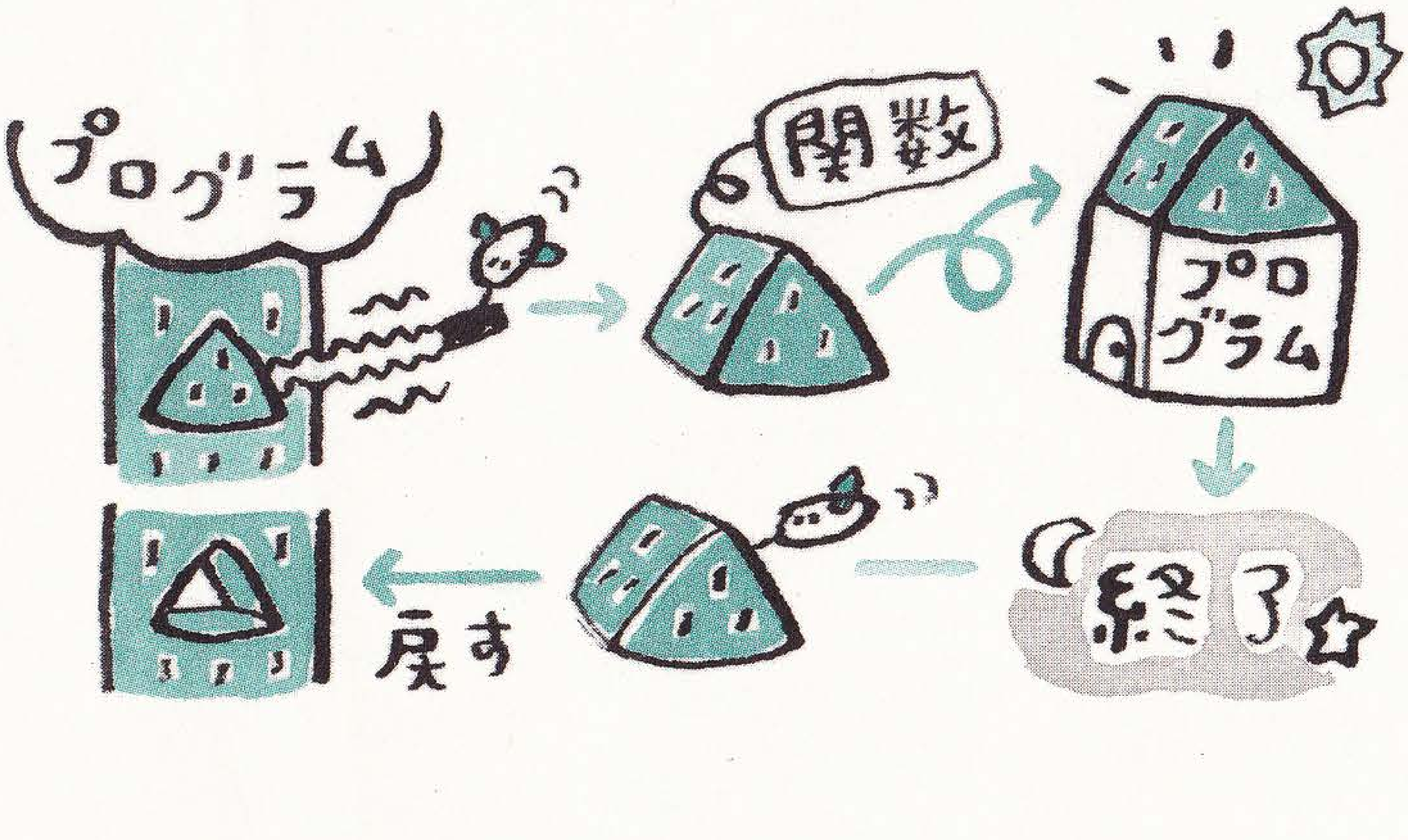
プログラムを起動することによって、このmain関数がはじめに動くしくみになっているんだ。


「それって、どういうことなの？」

特定のプログラムを起動する時は、そのプログラムのmain関数のみを動かすんだ。そのため、**main関数に他の関数を呼び出す処理を記述しておく**んだ。つまり関数は、プログラムに記述すべき処理を大まかにmain関数とそれ以外の関数に分けて記述し、**main関数がそれ以外の関数を順番に呼び出して処理する**ことによって、全体の処理を実現しているわけなんだ。

この関数を呼び出す処理を「**関数を利用する（P165参照）**」という。

関数は、プログラムの中で必要な部分を今のプログラムから切り分けて、別のプログラムに独立させたものだ。そして、プログラムの中から切り分けた部分をどこからでも自由に呼び出すことができるんだ。すると、その関数の処理内容を今動いているプログラムにつなげて処理することができる。
そして、関数の処理が終了すると、元のプログラムの場所に戻ってきて、何事もなかったかのように処理を続けることができる。
このように、プログラムは、main関数とその他の関数をうまく利用して成り立っているんだ。



たとえば、大きなプログラムを扱う場合を考えてみよう。
複数の人が同時にプログラムを作らなければならないような場合に、
それぞれの人が作成するプログラムの処理内容を整理して、
必要な時に部分的にそのプログラムを使った方が仕事の効率がいい。
だから、プログラムでは「**関数**」が重要なんだ。

このように関数は、プログラムを効率よく作る・動かすための機能といえる。

ポイント

- 関数は、main関数とその他の関数から成り立っている
- 関数は、main関数から別の関数を呼び出すことで利用することができる

関数を利用する

ここでは、関数を利用する方法について解説していこう。
関数を利用するには「**呼び出す**」作業が必要だったね。
呼び出す作業は、関数を定義し、呼び出すといった作業が必要になるんだけれど、算数プログラム（P138参照）にあるように、**標準ライブラリ（#include<stdio.h>）**には、特定のプログラムを呼び出す働きがある。このようにすでに「呼び出し」の準備がされている関数の場合は、特に自分で定義したり、呼び出す必要はない。

この標準ライブラリ関数というのは、あまりにも頻繁にプログラムで使われるために、標準で付けて使えるようにしてある関数で、特定のプログラムを呼び出すことが可能なんだ。

```
#include<stdio.h>                    //標準関数を使用するための宣言
```

「#include」は、「このプログラムでは、<stdio.h>で宣言した標準ライブラリ関数を使います」という宣言で、<stdio.h>というのは、標準入出力を制御するための関数のことを指しているんだ。

標準入出力は、パソコンのキーボードやディスプレイを制御するための関数群で、入出力制御の他にも、
ファイルアクセスをするための関数や、
エラー処理をするための関数が準備されている。

たとえば、標準入出力には以下のようなものがあるので覚えておこう。

・入出力制御　　　　scanf( )、printf( )、getchar( )、putchar( )、
・ファイルアクセス　fopen( )、fclose( )、fread( )、
・ファイル操作　　　remove( )、rename( )、
・エラー処理　　　　feof( )、ferror( )

実際に自分でプログラムを作るときには、上記のことを踏まえて
「関数の定義と呼び出し」をマスターしておくと便利だ。
順次、説明していこう。

関数を利用するには、

1. **関数を定義する**
2. **関数の宣言をする**（プロトタイプ宣言）
3. **関数を呼び出す**

の順にプログラムを作っていく。

まず、関数の定義からだね。
関数の定義では、
この関数では「**どんなデータを元にして**」「**どんな処理をするのか**」
を明確にすることが大切だ。
その際に、関数内部の処理内容や引数、戻り値などといった様々な要素が必要になってくるんだ。



「処理内容、引数、戻り値…。いっぱい要素が出てきて分からないなぁ。関数を作るときには、こんなに要素が必要なんですか？」

たとえば、パンを作るには、レシピ、パンをこねる機械、小麦粉やイースト菌、バター、牛乳などが必要で、これらの材料（要素）がなければパンは作ることができない。このパンづくりをプログラムにして考えてみよう。

全体の流れを**main関数**（パンを焼くための作業手順、材料が書かれている）とすると、この「こねるための機械」が**関数**（main関数に呼ばれる別の関数）ということになる。この処理を実際に行うのが人の手（**コンピュータ**）だ。

まず、人の手（コンピュータ）によって、材料（**引数**）がこねる機械（別の関数）に渡され、処理を施して、こねあがったパン生地を作り出す（**戻り値**）ということになるんだ。

そして、そのパン生地は、main関数に戻り、さらに、人の手（コンピュータ）でイースト菌（引数）を加えて、再びこねる機械（別の関数）で、もう一度こねる。

イースト菌も混ぜたパン生地（戻り値）は、また、人の手（コンピュータ）によって、発酵装置（別の関数）の中に入れる（引数）わけだ。発酵が終了すれば、また人の手（コンピュータ）によって、その次には、オーブン（別の関数）に入れて焼く。オーブン（別の関数）で焼いた後は、食べられるおいしいパン（戻り値）になって、パンができて、main関数が終了するというわけだ。

このように、関数はたくさんの要素から成り立っているということがわかるね。



では、関数に必要な要素を確認したところで、
算数プログラムを元に関数の定義を行ってみよう。
関数の定義というのは、どの処理を関数にするのかを考えることなので、プログラムをすべて書き出して、関数になりそうな部分に注目してみよう。

```
#include <stdio.h>          //標準関数を使用するための宣言
void main(){

        short int saishonosuuji;
        //変数saishonosuujiの宣言
        short int nibanmenosuuji;
        //変数nibanmenosuujiの宣言
        short int kekka;             //変数kekkaの宣言

        while(1){

                printf("最初の数字を入力して下さい ¥n");
                //最初のコメント表示
                scanf("%d",&saishonosuuji);
                //最初の変数を入力

                printf("2番目の数字を入力して下さい ¥n");
                //2つめのコメント表示
                scanf("%d",&nibanmenosuuji);
                //2つめの変数を入力

                kekka=saishonosuuji+nibanmenosuuji;
                //計算し、結果を変数kekkaへ入れる
```

元になるデータ

```
12              printf("結果:");
                //結果の表示
13              printf(" %d ", saishonosuuji);
14              printf(" + ");
15              printf(" %d ", nibanmenosuuji);
16              printf(" = ");
17              printf(" %d ", kekka);
18              printf("¥n");
        //結果の表示を終了し、改行

19          }
20  }
```

〜リスト4-1　算数プログラムより〜

まず「**どんなデータを元にするか**」に対しては、
11行目の「kekka = saishonosuuji + nibanmenosuuji;」の、右辺で足し算をし、その結果を左辺に代入する働きのあるこの一行に注目しよう。この時の、変数「saishonosuuji」と変数「nibanmenosuuji」が元になるデータだ。

次に「**どんな処理をするか**」だけれど、この一行を関数にする時、
関数は、main関数とその他の関数から成り立っている、とあったけれど（P162参照）、この時、main関数の変数saishonosuuji（引数）と変数nibanmenosuuji（引数）は、別の関数に渡す処理を行い（渡すデータを引数という，P179参照）、その別の関数で処理した結果をmain関数の変数kekka（戻り値）に取り込む処理（呼び出し側に戻す値を戻り値という，P182参照）を行う。

つまり、saishonosuuji+nibanmenosuujiを計算して、その結果を変数kekkaに伝える一連の処理のことを指しているんだ。

このように定義とは、
「どんなデータを元にするか」「どんな処理をするか」を明確にすることなんだ。
関数の処理内容や引数、戻り値が明確にできたので、
関数の定義の作業はできたことになる。

次に**関数の宣言**（**プロトタイプ宣言**）をしよう。
プロトタイプ宣言とは、**関数のデータ型の宣言**のことで、
関数を使用するための宣言のことをいうんだ。

宣言をする時は、変数の宣言の時のように、
データ型と関数名を決めることが大切だ。

まず、元になるコード（P169参照，リスト4-1）の03行目と04行目の宣言をみると変数を「short int」型で定義しているので、この2つの変数saishonosuujiとnibanmenosuujiのデータ型は同じく「short int」型としよう。

また、プロトタイプ宣言では、引数（saishonosuuji,nibanmenosuuji）が複数ある場合には、**個数分のデータ型の宣言を引数のデータ型として「,」で区切って書く**ので、これも理解しておこう。関数の名前は、処理内容がわかりやすいように、足し算の計算を行うということから「**tashizan**」としよう。

```
short int tashizan(short int,short int);
```

- short int → 型宣言
- tashizan → 関数名
- (short int,short int) → 引数のデータ型とその個数

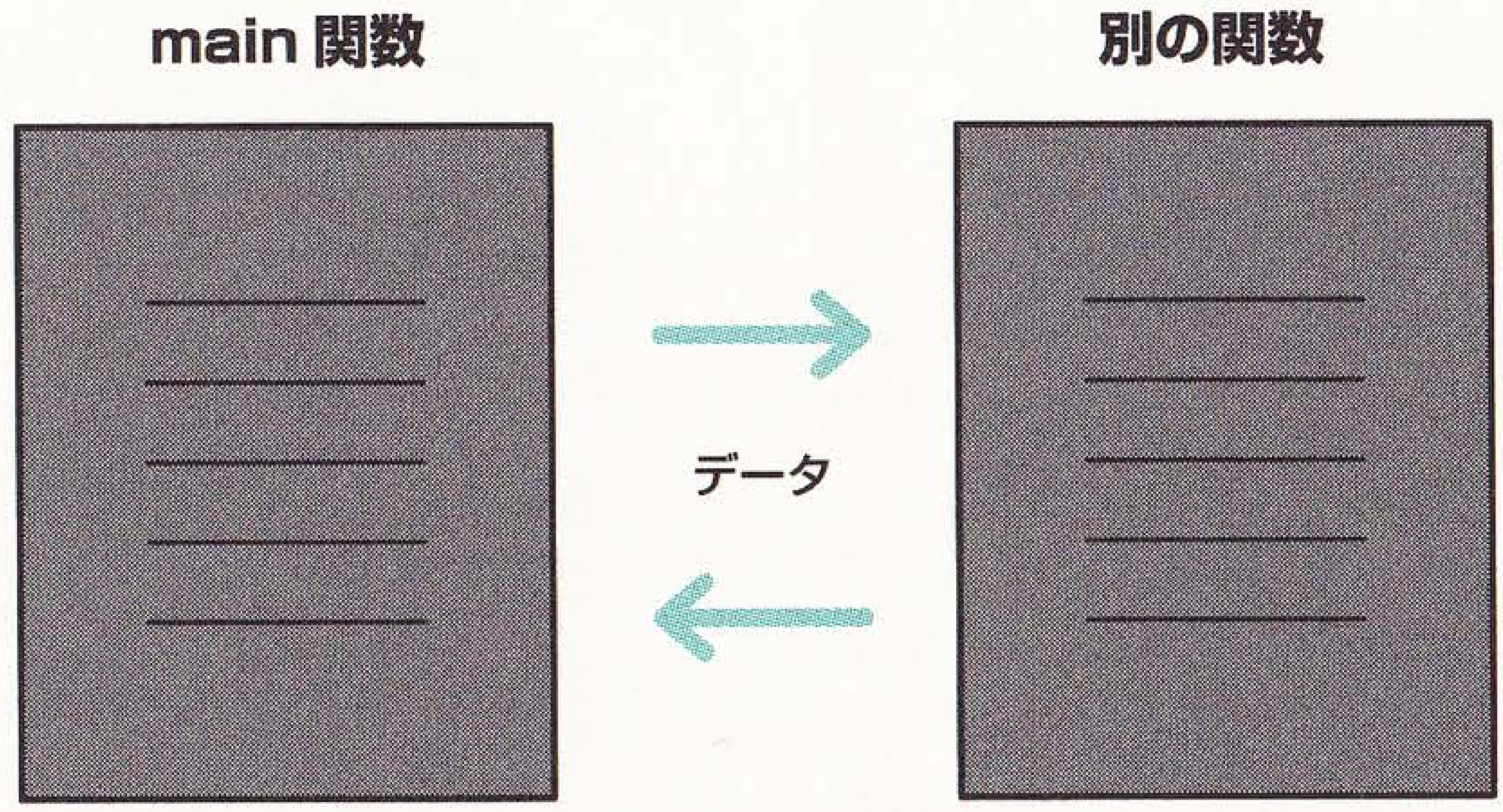
よって、void main ( ) { の前にshort int tashizan(short int, short int) ; を入れることでプロトタイプ宣言は完了する（P177参照, リスト4-2)。

プロトタイプ宣言

```
#include <stdio.h>
//標準関数を使用するための宣言
short int tashizan(short int, short int);
//関数tashizanを使用するための宣言
void main( ){
```

この時、プロトタイプ宣言では関数名の前にある「**short int**」は、**戻り値のデータ型**を示し、関数名の後にある「**(short int , short int)**」は、**引数のデータ型**と使う**引数の個数**を示している。

このことから、tashizanという名前の関数で、short int型のデータを2つ引数として使用する、ということを表わしているんだ。



●関数の処理

さて、関数を利用する時には、main関数から、別のプログラムへデータを渡して処理を行い、またmain関数へと戻る処理が必要だったね（P162参照）。
ここで、main関数のプログラムから別のプログラムを呼び出す方法についてみていこう。

関数の呼び出しは、「関数名」と「引数の変数名」を書くだけでいい。main関数のwhileループ（P177参照，リスト4-2の07行目～19行目）の中で、

関数の呼び出し

```
12  kekka=tashizan(saishonosuuji,nibanmenosuuji);
    //関数呼び出し
```

と書くことで、**関数を呼び出す**ことになる。ここで実際に、関数を呼び出しているのは、tashizan(saishonosuuji,nibanmenosuuji) の部分だ。

この2つの変数は、main関数から別のプログラムを呼び出す時の、呼び出される側に渡すデータのことをいい、**引数**と呼ばれるものなんだ。

 **「この一行だけで、意外と簡単に呼び出せるね。」**

一方、呼び出される側の関数は、どこに書くのかというと、main関数の後に続けて書くんだ（P178参照,リスト4-2）。

呼び出される関数

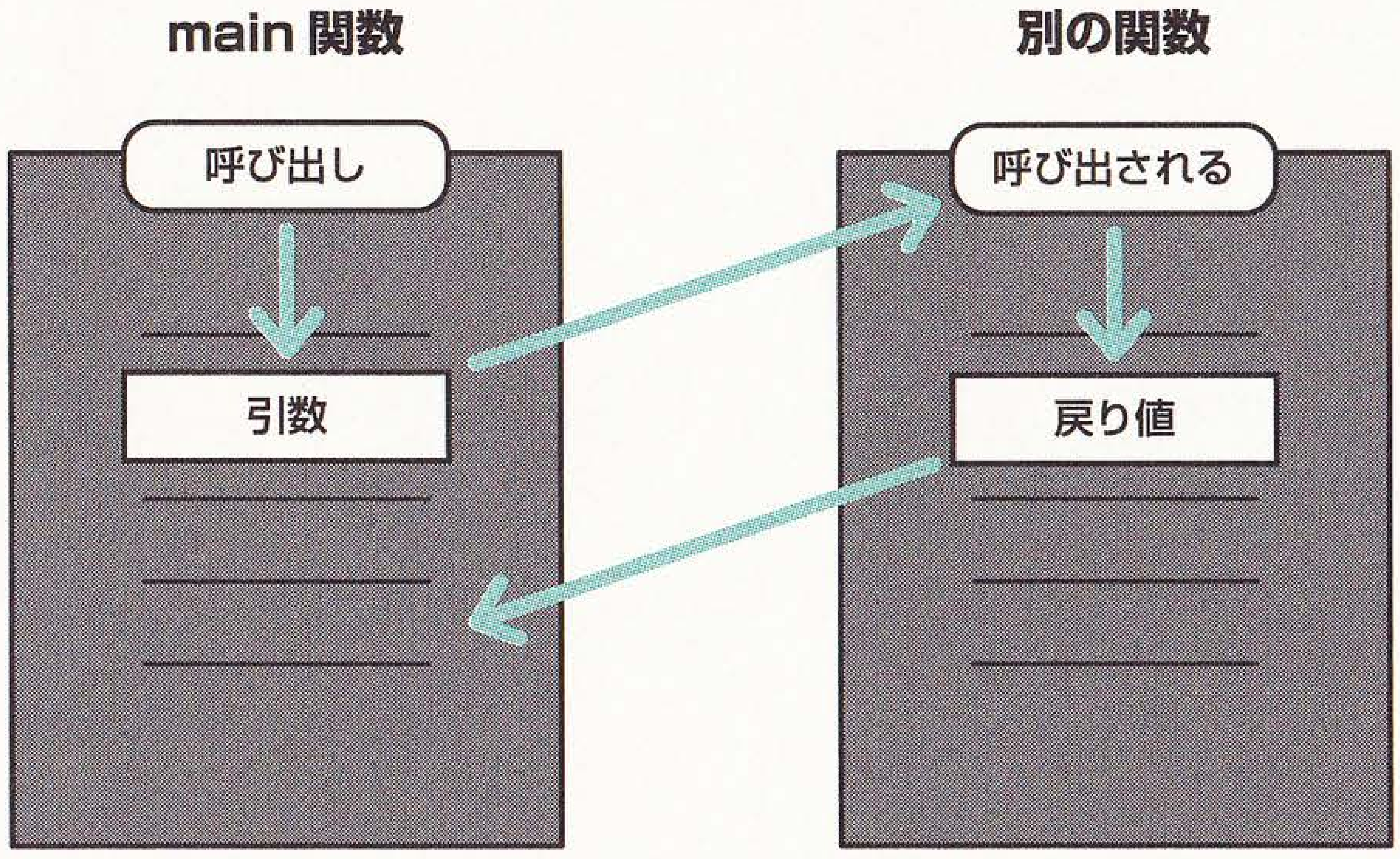
```
22  short int tashizan(short int first
   ,short int second){
```

関数は呼び出されると、main関数が動き始めるのと同じように、この関数の処理を進めていくんだ。

ここで、プロトタイプ宣言との違いは、引数の項が**データ型**と具体的な**変数名**になっているということだ。
また、12行目の呼び出す側のmain関数tashizanとは、変数名が変わっていることがわかるね。
これは計算結果を出す時の、関数tashizanの中だけで使用する**ローカル変数**なので、関数を呼び出す側のmain関数と使う変数を混合しないためなんだ。

この後、ここでのローカル変数の関数は、1番目の数字と2番目の数字を計算して、結果を出している。

```
short int answer;
answer = first + second;
```



● 関数の呼び出し

最後に、計算結果をmainプログラムに戻り値として渡す関数として、

```
25          return answer;    //戻り値はanswer
26  }
```

と記述する。これは、元のプログラムに戻れということと、その時に、変数answerの内容を戻り値にすることを示している。

return answer ;の後、プログラムは、main関数の呼び出しを行った12行目kekka=tashizan（saishonosuuji,nibanmenosuuji）；に進んで戻り値「**answer**」を「**kekka**」に渡し処理を継続することになる。これで**呼び出された関数の処理は終了**し、**呼び出した方のmain関数は次の行に処理を進めていく**んだ。

このように関数を利用する時には、
「**関数を定義する**」「**関数の宣言をする（プロトタイプ宣言）**」「**関数を呼び出す**」の順に作っていくのだけれど、関数を作る時には、どんな処理を関数にするのか考えて、呼び出す方のプログラムとどんなデータをやりとりするのかを考える。
その後で、呼び出す方のプログラムで関数のプロトタイプ宣言をしてから、関数のプログラムを作成していけばいいんだ。

リスト4-2　関数の利用

```
#include <stdio.h>        //標準関数を使用するための宣言
short int tashizan(short int, short int);
//関数tashizanを使用するための宣言
void main(){
        short int saishonosuuji;
        //変数saishonosuujiの宣言
        short int nibanmenosuuji;
        //変数nibanmenosuujiの宣言
        short int kekka; //変数kekkaの宣言
        while(1){
                printf("最初の数字を入力してください ¥n");
                //最初のコメント表示
                scanf("%d",&saishonosuuji);
                //最初の変数を入力
                printf("2番目の数字を入力してください ¥n");
                //2つめのコメント表示
                scanf("%d",&nibanmenosuuji);
                //2つめの変数を入力
    kekka=tashizan(saishonosuuji,nibanmenosuuji);
    //関数呼び出し
        printf("結果:");     //結果の表示
        printf(" %d ", saishonosuuji);
        printf(" + ");
        printf(" %d ", nibanmenosuuji);
        printf(" = ");
        printf(" %d ", kekka);
```

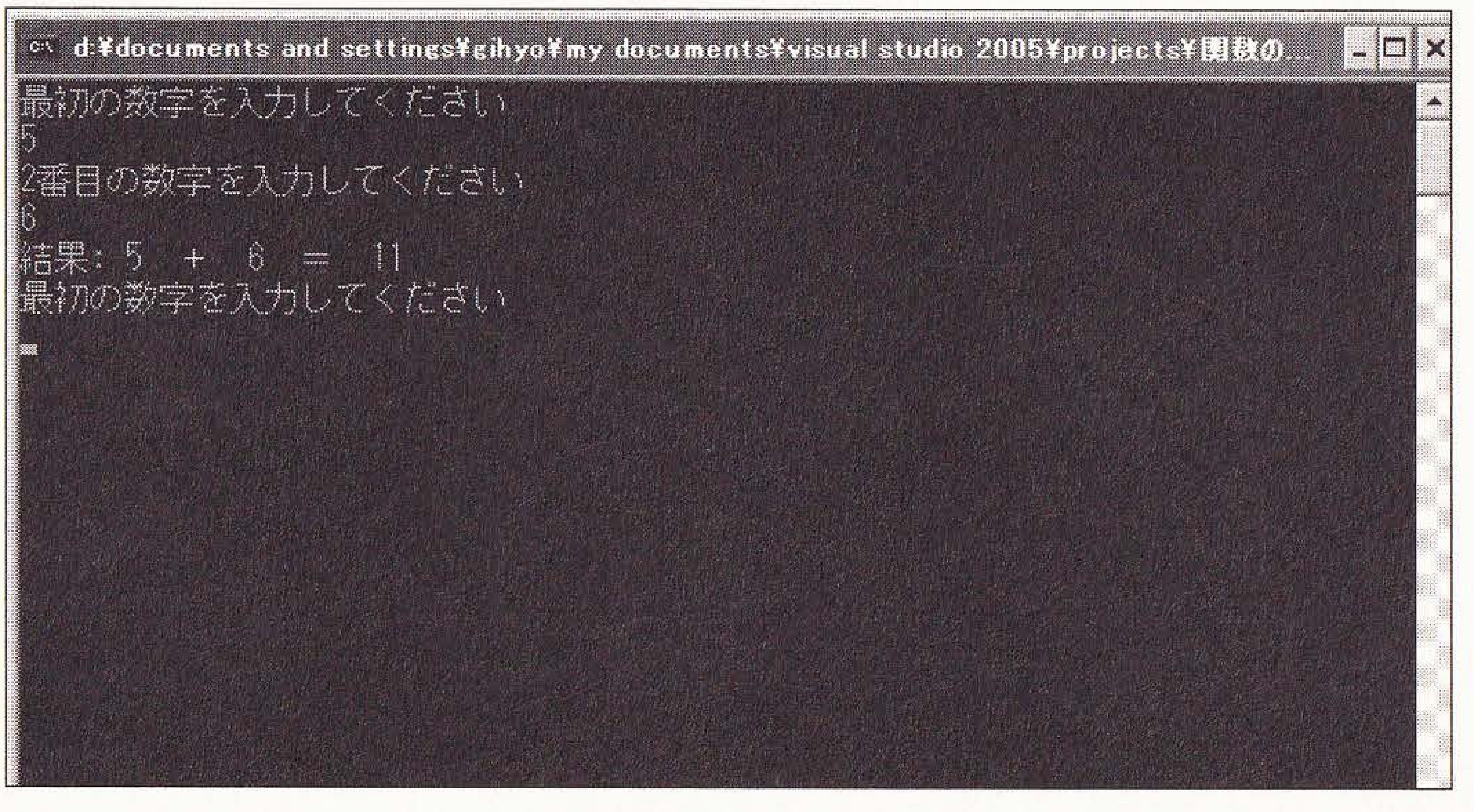
プロトタイプ宣言

呼び出しを行う関数

```
        printf("¥n");        //結果の表示を終了し、改行
        }
}
```

呼び出される関数

```
        short int tashizan(short int first,short
int second){
        short int answer;                //変数answerの宣言
        answer = first + second;
        //計算し、結果を変数answerへ入れる
        return answer;  //戻り値はanswer
}
```



● 関数の利用

ポイント

関数を利用するときには、「関数を定義する」「関数の宣言をする（プロトタイプ宣言）」「関数を呼び出す」ことが必要である

引数

関数についてうまく理解できたら、引数についておさらいをしておこう。
引数は、main関数から別の関数を呼び出して処理を行う場合に、
関数を呼び出す側のプログラム（main関数）から、
呼び出される側の関数に渡すデータのことを指していたね。
関数を呼び出す時に、関数名の後の（ ）の中に入るのが引数で、
変数saishonosuujiと変数nibanmenosuujiを引数として示しているよ。

```
kekka=tashizan(saishonosuuji,nibanmenosuuji);
//関数呼び出し
```

引数（saishonosuuji,nibanmenosuuji）

関数で引数を使うときには、

① 関数のプロトタイプ宣言で引数を使うということを定義する
② 関数の処理の中で受け取った引数を扱う

ことがポイントだ。

では、引数の定義から確認してみよう（P177参照，リスト4-2）。

```
    short int tashizan(short int,short int)
```

この関数tashizanを使う宣言では、関数tashizanを使うには、
2つのshort int型の引数が2つ必要だということを示している。

これによって、main関数から関数tashizanを呼び出す時には、関数名の後の（ ）の中に、short int型の引数を2つ入れる、ということがわかる。

そして、あらかじめmain関数の

```
04        short int saishonosuuji;
05        short int nibanmenosuuji;
```

で、short int型の変数を定義してあるので、呼び出す時には、
この変数を直接書き込んでいる。

では、関数を呼び出す部分を確認してみよう。
実際に関数を呼び出しているのは、main関数の12行目だ。

```
12    kekka=tashizan(saishonosuuji,nibanmenosuuji);
```

変数saishonosuujiと変数nibanmenosuujiは、
22行目の関数tashizanへ引数として渡されることを表している。

引数として渡す側、つまり呼び出される側の関数を確認してみよう。

```
22   short int tashizan(short int first
     ,short int second){
```

12行目の呼び出し側から、この22行目に情報が与えられるんだ。



「あれれ？　引数は、saishonosuujiとnibanmenosuujiではなかったの？」

そうだったね。main関数から呼び出す時には、

```
kekka=tashizan(saishonosuuji,nibanmenosuuji);
//関数呼び出し
```

と示していたよね。つまり、変数saishonosuujiとnibanmenosuujiを入れたんだけれど、関数を呼び出す時の引数は、呼び出す側と呼び出される側で異なる変数として扱うんだ。こうすることで、呼び出す側と呼び出される側をプログラマが混合しないですむんだ（P174参照）。関数名と引数のデータ型さえ合っていれば、呼び出された関数はうまく働くので問題はないよ。

ポイント

- 引数を使う時には、「関数のプロトタイプ宣言で引数を使うということを定義する」「関数の処理の中で引数を扱う」ことが必要である
- 関数を呼び出す場合、呼び出される関数の関数名とデータ型は、呼び出す側の関数と同じもので示すが、変数は呼び出す側と呼び出される側では、異なる変数として扱う

戻り値

戻り値は関数を呼んだ時に呼ばれた側の関数から、
呼び出す側のプログラムに渡すデータのことをいう。

戻り値を利用するには、
① 関数の定義で戻り値の型を定義する
② returnという文を使って値を返す
がポイントになる。

さて、算数プログラムの関数tashizanでは、プロトタイプ宣言で戻り値short intを指定していたね（P177参照，リスト4-2）。

```
02   short int tashizan(short int, short int);
     //関数tashizanを使用するための宣言
```

02行目で関数tashizanはshort int型の戻り値を渡すということを宣言しているんだ。
そのため、呼び出される関数もshort int型のデータ型を記述するということが大切なんだ。

```
22   short int tashizan(short int first
     ,short int second){
```

次にreturn文を使って値を返す、というところだ。
戻り値は、関数プログラムの最後にあるreturn文に続けて書いてある、変数answerだ。

戻り値

```
25        return answer;   //戻り値はanswer
```

変数answerには計算結果が入っているので、上記のように書くことによって、変数answerの値は呼び出した方のmain関数プログラムの12行目に渡されることを示している。

```
12        kekka=tashizan(saishonosuuji,nibanmenosuuji);
          //関数呼び出し
```

このように関数tashizanからの戻り値「answer」は、変数「kekka」に渡されることがわかるね。
この後、mainプログラムは、変数kekkaの値を表示することができる、というわけだ。

ポイント

戻り値を利用するには「戻り値の型を定義する」「returnという文を使って値を返す」ということが必要である

配列とポインター

算数プログラムでは、変数をたくさん宣言して利用したね（P138参照）。

でも、足し算プログラムで使っている変数が、もっともっと多くなったら、大変なことになる。


「もっともっと多くなるって、どういうことですか？」

たとえば、足し算の数が1＋2＋3＋4＋5＋6＋‥‥と多くなったら、変数は、「saishonosuuji」「nibanmenosuuji」「sanbanmenosuuji」「yonbanmenosuuji」といった具合に増やす必要があるかもしれない。

そんなときに**配列**が便利なんだ。


「配列って？」

配列というのは、同じ型の変数をまとめて宣言するもので、

これを使えば一気に必要な数の変数を宣言できる。

それに、変数を1つひとつ宣言するより、データの数や大きさがわかりやすくなるんだよ。


「配列ってどうやって使うものなの？」

配列も変数と同じように、宣言が必要で

型や配列名を用意することが大切だ。

そして、値がいくつ必要かを指定するんだ。

これを**配列の要素**という。

足し算プログラムでいえば、変数saishonosuujiとか変数nibanmeno-suujiというのが出てくるけど、これを配列名suujiとして宣言してみよう。

```
short int suuji[4] ;
```

このように、足す数（変数）が4つある場合の宣言をしたときは、

```
suuji[0]
suuji[1]
suuji[2]
suuji[3]
```

という4つの同じshort int型のデータとして扱うことができるんだ。変数に当てはめると、それぞれが変数名のようなものになるね。



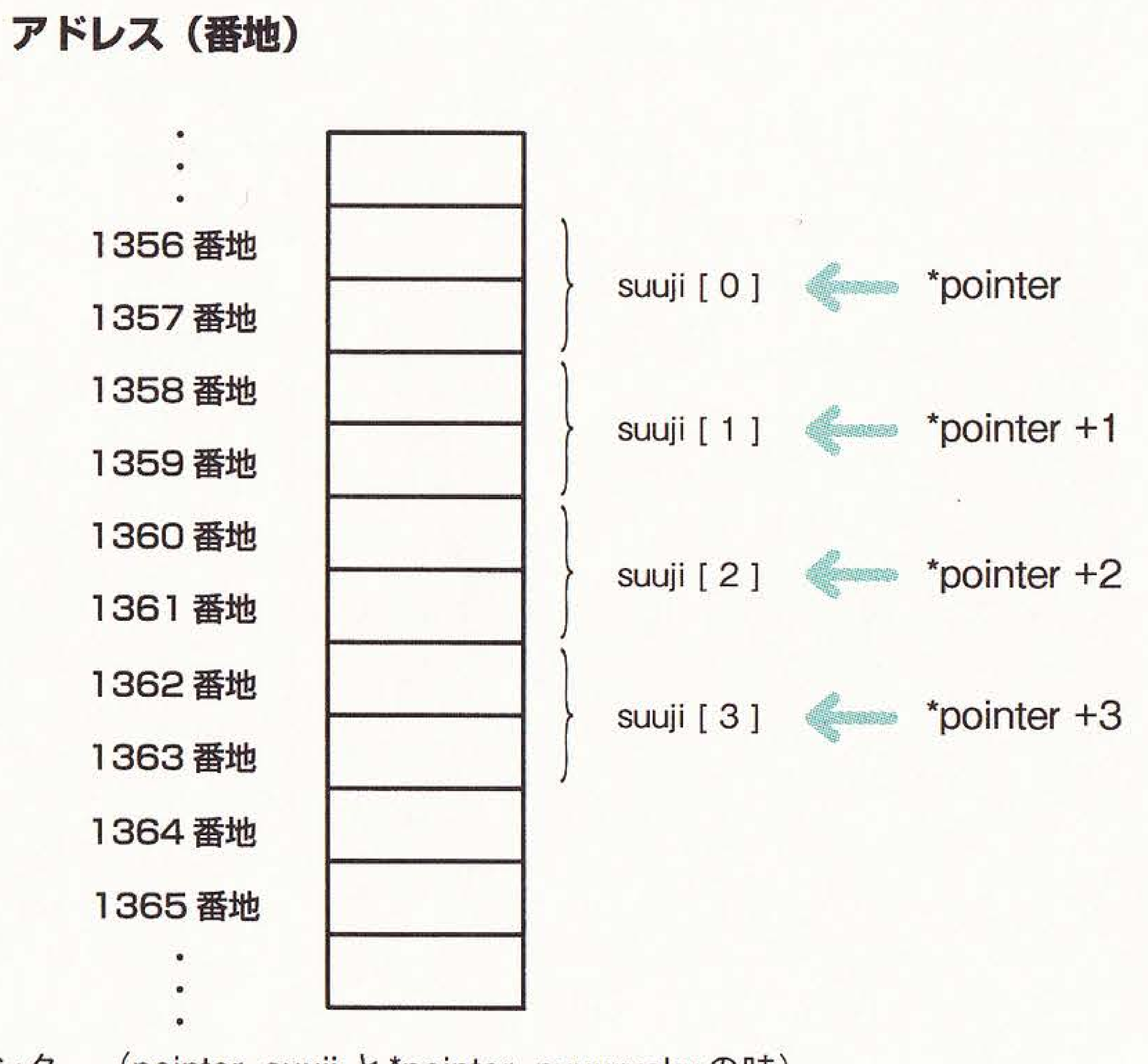
● 配列

プログラムの中では、配列のどの要素を使うのかがわからないといけない。ここでキー入力される4つの数字を順番に配列に入れることが

大切で、最初の入力をsuuji[0]、2番目をsuuji[1]、3番目をsuuji[2]、4番目をsuuji[3]として表わすことができる。

このように宣言した配列の特定の要素を見分けるために使うのが、**ポインター**だ。ポインターは配列の宣言を参照して、配列の中の特定要素を指し示す役割がある。
ポインターを使う時には変数の前に"*"をつけて、変数とは区別するんだ。

たとえば、*pointerはsuuji[0]、*pointer+1はsuuji[1]、*pointer+2はsuuji[2]、*pointer+3はsuuji[3]を参照し、配列の特定要素を指し示すことができるんだ。



● 配列とポインター（pointer=suuji と *pointer=nyuuryoku の時）

実際に、足し算プログラムをもとに配列とポインターが利用されているプログラムをみてみよう（P194参照，リスト4-3）。

```
#include <stdio.h>      //標準関数を使用するための宣言
void main(){
```

ここからがmain関数の始まりだ。
まず、配列の宣言からだ。

```
short int suuji[4];      //配列suujiの宣言
```

この宣言ではshort int型の配列として、
suuji[0]、suuji[1]、suuji[2]、suuji[3]の4つの要素が定義できる。
これらは、もとのプログラムでは変数として宣言していたんだけど、同じデータ型の4つの変数になるので、配列でまとめて宣言していて、変数saisyonosuujiや変数nibanmenosuujiと同じように使えるんだ。

```
short int *pointer;
//ポインター変数pointerの宣言
```

これは、配列の要素を決めるためのポインターを変数として宣言しているんだ。データ型はshort intで宣言している。また、このポインターは配列suujiの要素を扱うので、取り得る値は、4つになる。

```
short int nyuuryoku;
//変数nyuuryokuの宣言
```

ここでは、キー入力された値を一時的に記憶するための変数の定義をする。キー入力は、この変数に一度記憶してから、配列の要素にコピーすることを表しているんだ。

```
06    short int kekka;                //変数kekkaの宣言
```

この変数は、計算結果を記憶する。もとのプログラムにもあったよね。

```
07        while(1){
```

07行目からいよいよプログラムの始まりだ。

```
08            pointer = suuji;
               //ポインター変数に配列の先頭アドレスを設定
```

08行目では、ポインター変数のpointerに配列suujiの先頭アドレスのデータを入れることを設定している。このようにすることでポインターは、配列suujiの要素を指し示すことができるようになるんだ。



「変数pointerとポインター変数*pointerの違いって何？」

pointerは、変数そのものを表し、*pointerは、変数pointerに入れた数値データで示すところのアドレスに記憶されているデータを示しているんだ。

```
09            printf("最初の数字を入力して下さい ¥n");
                                  //最初のコメント表示
```

これは、もとのプログラムと同じく出力を表し、最初の数字のキー入力を促しているんだ。

```
10        scanf("%d",&nyuuryoku);     //最初の変数を入力
```

上記では、キー入力された数値データを変数nyuuryokuに記憶することを示しているよ。

```
11        *pointer = nyuuryoku;
          //入力された数字を配列に入れる
```

11行目では、変数nyuuryokuの記憶された数値を配列の要素にコピーすることを表している。ここでは、配列の要素である（suuji[0]）をポインターが指し示し、この配列要素に変数nyuuryokuの値をコピーしていることになる。

```
12        printf("2番目の数字を入力して下さい ¥n");
          //2番目のコメント表示
```

これは、もとのプログラムと同じく、2番目の数字のキー入力を促す出力を表すよ。

```
13        scanf("%d",&nyuuryoku);
          //2番目の変数を入力
```

これは、2番目にキー入力された数値データを再び変数nyuuryokuに記憶するという意味だ。

```
14          *(pointer +1) = nyuuryoku;
             //入力された数字を配列に入れる
```

14行目では、変数nyuuryokuの数値を配列の要素にコピーする。ここでは、pointer+1つまり、ポインターは配列の次の要素（suuji[1]）を指し示し、変数nyuuryokuの値をコピーしているということなんだ。

```
15          printf("3番目の数字を入力して下さい ¥n");
            //3番目のコメント表示
```

これも、3番目の数字のキー入力を促す出力を表している。

```
16        scanf("%d",&nyuuryoku);    //3番目の変数を入力
```

16行目では、3番目にキー入力された数値データを、再び変数nyuuryokuに記憶することを示している。

```
17          *(pointer + 2) = nyuuryoku;
             //入力された数字を配列に入れる
```

これは、変数nyuuryokuの数値を配列の要素にコピーする。ここでは、pointer+2つまり、ポインターは配列の次の、次の要素（suuji[2]）を指し示し、変数nyuuryokuの値をコピーしているということなんだ。

```
18          printf("4番目の数字を入力して下さい ¥n");
            //4番目のコメント表示
```

これも、4番目の数字のキー入力を促す出力を表している。

```
19          scanf("%d",&nyuuryoku);   //4番目の変数を入力
```

ここでは、4番目にキー入力された数値データを、再び変数nyuuryokuに記憶している。

```
20          *(pointer + 3) = nyuuryoku;
             //入力された数字を配列に入れる
```

20行目は、変数nyuuryokuの数値を配列の要素にコピーする。ここでは、pointer+3つまり、ポインターは配列の次の、次の、次の要素(suuji[3])を指し示し、変数nyuuryokuの値をコピーしているということなんだ。

```
21          kekka=suuji[0]+suuji[1]+suuji[2]+suuji[3];
            //計算し、結果を変数kekkaへ入れる
```

ここでは、これまでキー入力してきた数値の計算を行う。これらの数値は、すべての配列の要素にコピーしてきたので、4つの数値の足し算をして変数kekkaにその計算結果を代入する、ということを表しているんだ。

```
22          printf("結果:");              //結果の表示
```

ここからは、計算の内容を表示するプログラムだ。

```
23        printf(" %d ", *(pointer++));
          //suuji[0]に入っている数字を表示してpointerをイン
    クリメント
```

23行目の*(pointer++)は、pointerで示される配列の要素のデータを示している。このように書くことで、配列要素suuji[0]の内容が表示される。その後に、++と記述しているけれど、これは、表示の処理が終わってから、変数pointerを＋1する、ということを示している。

```
24        printf(" + ");
```

これは、もとのプログラムと同じく足し算の演算子である「＋」を表示する。

```
25        printf(" %d ", *(pointer++));
          //suuji[1]に入っている数字を表示してpointerをイン
    クリメント
```

ここでは、変数pointerの値は、23行目で＋1されたことにより、次の配列要素suuji[1]が指定されていることになる。つまり、表示されるのは、配列要素suuji[1]の内容だ。

```
26        printf(" + ");
```

これも、変数の足し算の出力を表している。

```
27          printf(" %d ", *(pointer++));
            //suuji[2]に入っている数字を表示してpointerをイン
    クリメント
```

ここでは、変数pointerの値は、25行目で＋1されたことにより、次の配列要素suuji[2]が指定されていることになる。表示されるのは、配列要素suuji[2]の内容だ。

```
28          printf(" + ");
```

これも、同じように、変数の足し算の出力を表している。

```
29          printf(" %d ", *(pointer++));
            //suuji[3]に入っている数字を表示してpointerをイン
    クリメント
```

ここでは、変数pointerの値は、27行目で＋1されたことにより、次の配列要素suuji[3]が指定されていることになる。表示されるのは、配列要素suuji[3]の内容だ。

```
30          printf(" = ");
31          printf(" %d ", kekka);      //結果を表示する
32          printf("¥n");              //改行する
33          }
34  }
```

最後に30～34行目は、計算結果を表示することを表している。

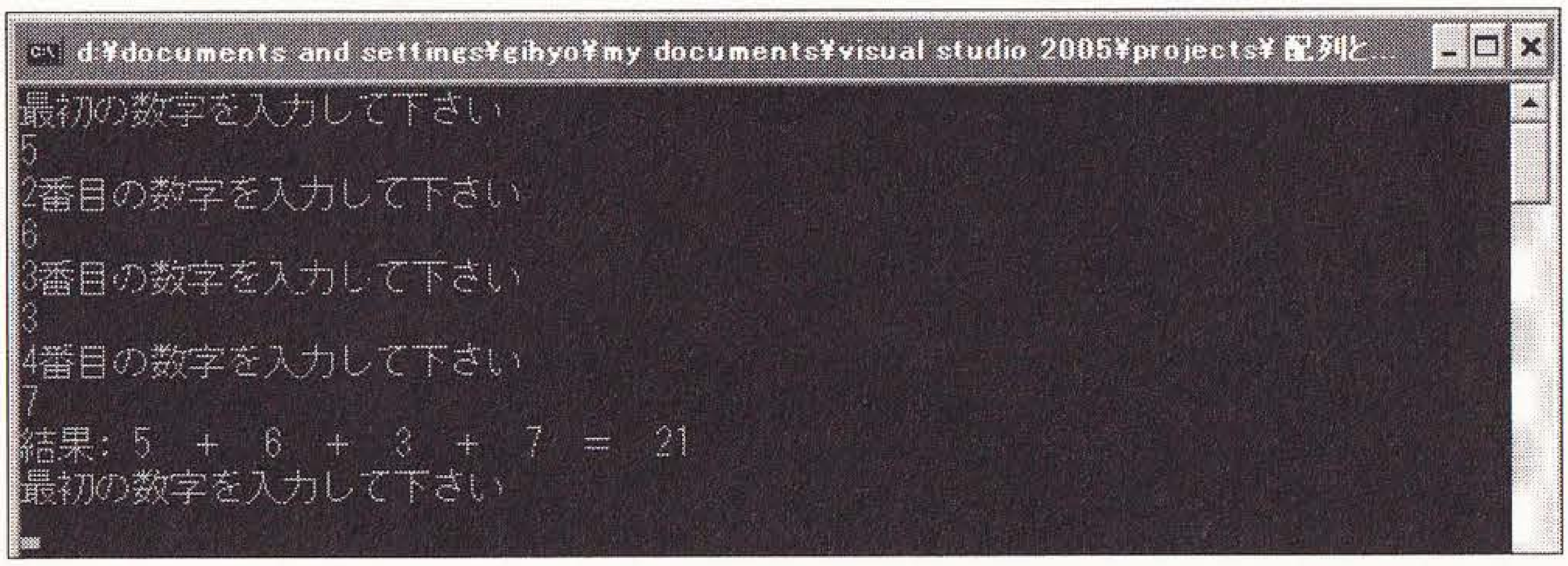
このように変数の数が多い場合のプログラムでは、配列とポインター

を利用することで、配列の特定の要素を指し示し、数値データを入力したり、その内容を表示させることができるとても便利な機能なんだ。

リスト4-3　算数プログラムの配列とポインターの利用

```
#include <stdio.h>        //標準関数を使用するための宣言
void main(){
short int suuji[4];       //配列suujiの宣言
short int *pointer;
//ポインター変数pointerの宣言
short int nyuuryoku;
//変数nyuuryokuの宣言
short int kekka;
//変数kekkaの宣言
  while(1){
    pointer=suuji;
    //ポインター変数に配列の先頭アドレスを設定
    printf("最初の数字を入力して下さい ¥n");
                      //最初のコメント表示
    scanf("%d",&nyuuryoku);
    //最初の変数を入力
    *pointer = nyuuryoku;
    //入力された数字を配列に入れる
    printf("2番目の数字を入力して下さい ¥n");
    //2番目のコメント表示
    scanf("%d",&nyuuryoku);  //2番目の変数を入力
    *(pointer +1) = nyuuryoku;
    //入力された数字を配列に入れる
    printf("3番目の数字を入力して下さい ¥n");
    //3番目のコメント表示
    scanf("%d",&nyuuryoku);  //3番目の変数を入力
```

```
        *(pointer + 2) = nyuuryoku;
        //入力された数字を配列に入れる
        printf("4番目の数字を入力して下さい ¥n");
        //4番目のコメント表示
        scanf("%d",&nyuuryoku);  //4番目の変数を入力
        *(pointer + 3) = nyuuryoku;
        //入力された数字を配列に入れる
        kekka=suuji[0]+suuji[1]+suuji[2]+
suuji[3];
        //計算し、結果を変数kekkaへ入れる
        printf("結果:");           //結果の表示
         printf(" %d ", *(pointer++));
        //suuji[0]に入っている数字を表示してpointerをイン
クリメント
        printf(" + ");
        printf(" %d ", *(pointer++));
        //suuji[1]に入っている数字を表示してpointerをイン
クリメント
        printf(" + ");
        printf(" %d ", *(pointer++));
        //suuji[2]に入っている数字を表示してpointerをイン
クリメント
        printf(" + ");
        printf(" %d ", *(pointer++));
        //suuji[3]に入っている数字を表示してpointerをイン
クリメント
        printf(" = ");
        printf(" %d ", kekka);      //結果を表示する
        printf("¥n");               //改行する
        }
}
```



● 配列とポインター

配列やポインターの考え方は、多くの人が最もつまずきやすいところだが、C言語やC++言語でプログラムを作成するときにはとても重要で、役に立つ考え方なんだ。

ポイント

- 配列は、同じ型の変数をまとめて宣言するものである
- ポインターは宣言した配列の特定の要素を見分けるために使う

まとめ

- データは、メモリ上の変数に一時記憶され、必要なときに読み出されるこの時、アドレスは変数の場所を特定する働きを行っている
- 関数は、変数の値と数値を比較して処理を変えるしくみがある。このしくみを条件分岐といい、条件分岐を行うプログラムを条件文という
- 関数は、プログラムに記述すべき処理をmain関数とそれ以外の関数に分けて、必要な時にmain関数から別の関数を呼び出して処理を行っている
- 関数を利用するには、「関数を定義する」「関数の宣言をする（プロトタイプ宣言）」「関数を呼び出す」ことが必要である
- 関数を動かすために、main関数から呼び出す「引数」と別のプログラムからmain関数に戻す「戻り値」がある
- たくさんの変数を利用するときは、まとめて一気に必要な数の変数を宣言できる「配列」と、その1つひとつを指し示す「ポインター」を使うことで効率よく処理することができる

第5章
プログラムの完成



算数選手権が近づき、少し焦ってきたくま。
絵美のお小遣い計算の勉強に付き合って、少しずつ算数とコンピュータについてはわかってきてはいるが、算数選手権にはコンピュータを持ち込んではいけないことがわかった。

くま「はあ～。どうしよう。」

冷や汗をかくくま…。

絵美は、先生にくま君の算数の実力を上げる手段はないか、
と親身になって相談する。

絵美やくまも、せっかくプログラミングの基本が分かってきたので、
これはよい傾向と考えた先生は、くま君が算数選手権までに
パソコンで算数を特訓するプログラムがあったらどうだろう、
と考えた。

先生「くま君、みんなでプログラムを作って、算数選手権の優勝を狙ってみないか？」

くま「そのプログラムを使って特訓すれば優勝できる？」

まったく、他力本願な性格は直っていないらしい。

先生「保証はできないけれど、やってみる価値はあるんじゃない？」

絵美「そうだよ。みんなで算数選手権に向けて頑張ろう！」

こうして、くま君の算数の実力を上げるべく、算数特訓プログラムを作ることとなった。



5-1 プログラムの作成

分析〜何を作るのか〜

算数選手権が近づいてきている。
今までの勉強をもとに、実際に算数特訓プログラムを作ってみよう。



「特訓って、計算を間違えたら、お尻を叩かれたりするんですか？　ちょっと、イヤだなぁ・・・。」

そんなに厳しいものでもないよ。
くま君が算数の実力をつけるためにプログラムを作って練習をしていこう。ただ、それだけのことだよ。

まずは、パソコンに何をさせるかを考える**分析**からだ。
これは、プログラミングの最初にすることだったね（P80参照）。

プログラミングと分析は、全然違うような気がするけど、
実際、プログラムを作る前に、何をするのかを決めておかないと、
プログラムはうまく作ることができない。

くま君は、パソコンで算数の計算を練習するって、どんなイメージを持っている？

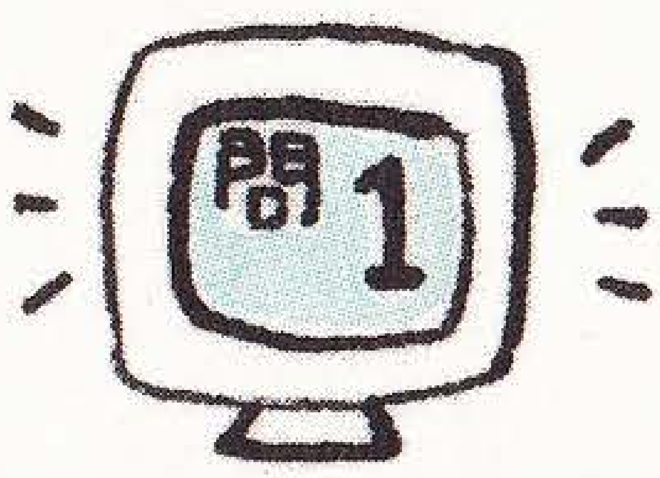


「そうだな～。パソコンに、計算式が表示されて…。う～ん、その後は…。」

それではまず算数のプログラムのイメージを整理してみよう。
パソコンはすでに起動している状態にある。
くま君が算数特訓プログラムを動かすところから順番に書いてみよう。



①くま
プログラムを動かす。



②パソコン
プログラムの最初の問題を表示する。



③くま
表示された問題を解いて、答えをキー入力する。



④パソコン
キー入力を受け付けて、入力されたデータが問題の計算結果と合うか確認する。合っていたら、次の問題を表示する。

「すごいすごい、次から次へと問題を解くのね。
特訓になるね。」

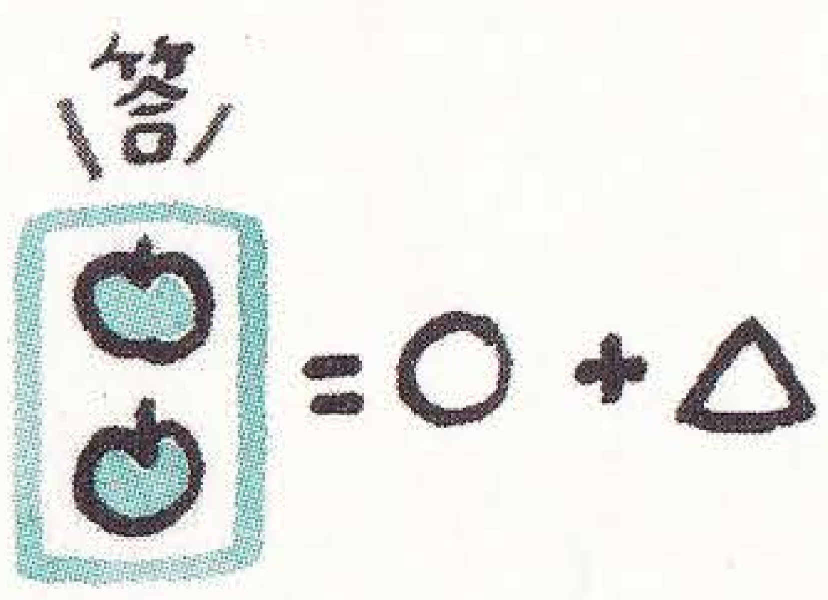
「ちょっと、待ってよ。
間違った答えを入れたら、どうなるの？」

いいところに気がついたね。
間違った場合は、どうしたらいいかな。

「やっぱり、同じ問題をもう一度やったほうがいいと思います。」

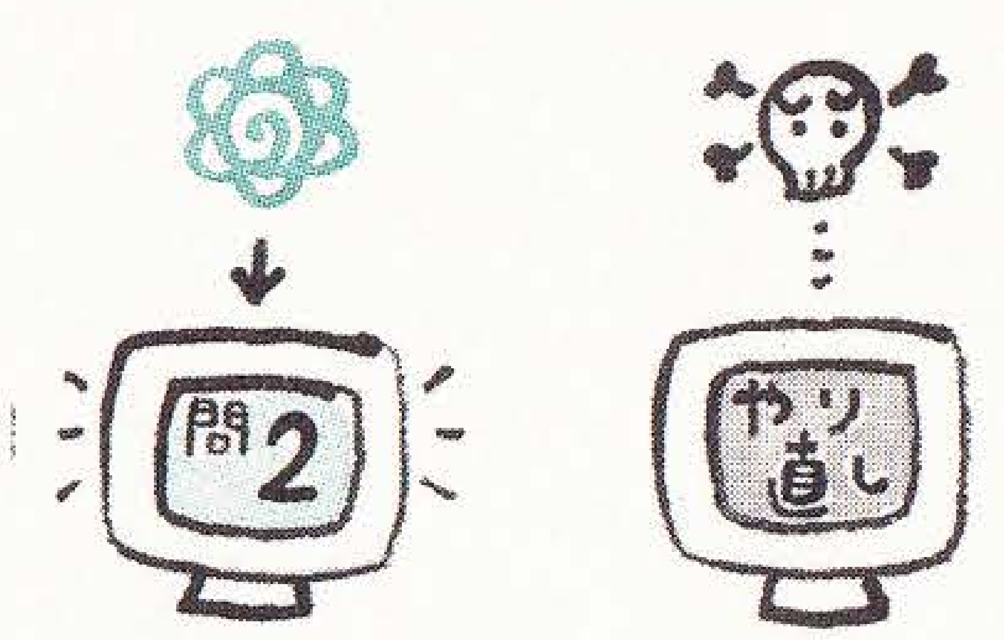
それでは、間違った場合には同じ問題をもう一度やり直すことにしよう。
先ほど考えた「パソコンの動き」の最後の④のところを、

④パソコン
キー入力を受け付けて、入力されたデータが
問題の計算結果と合っているかを確認する。



→合っていたら、次の問題を表示する。
→間違っていたら、同じ問題を表示する。

としよう。



これで、1問目はOKだね。

2問目以降は、①と②はいらないので、

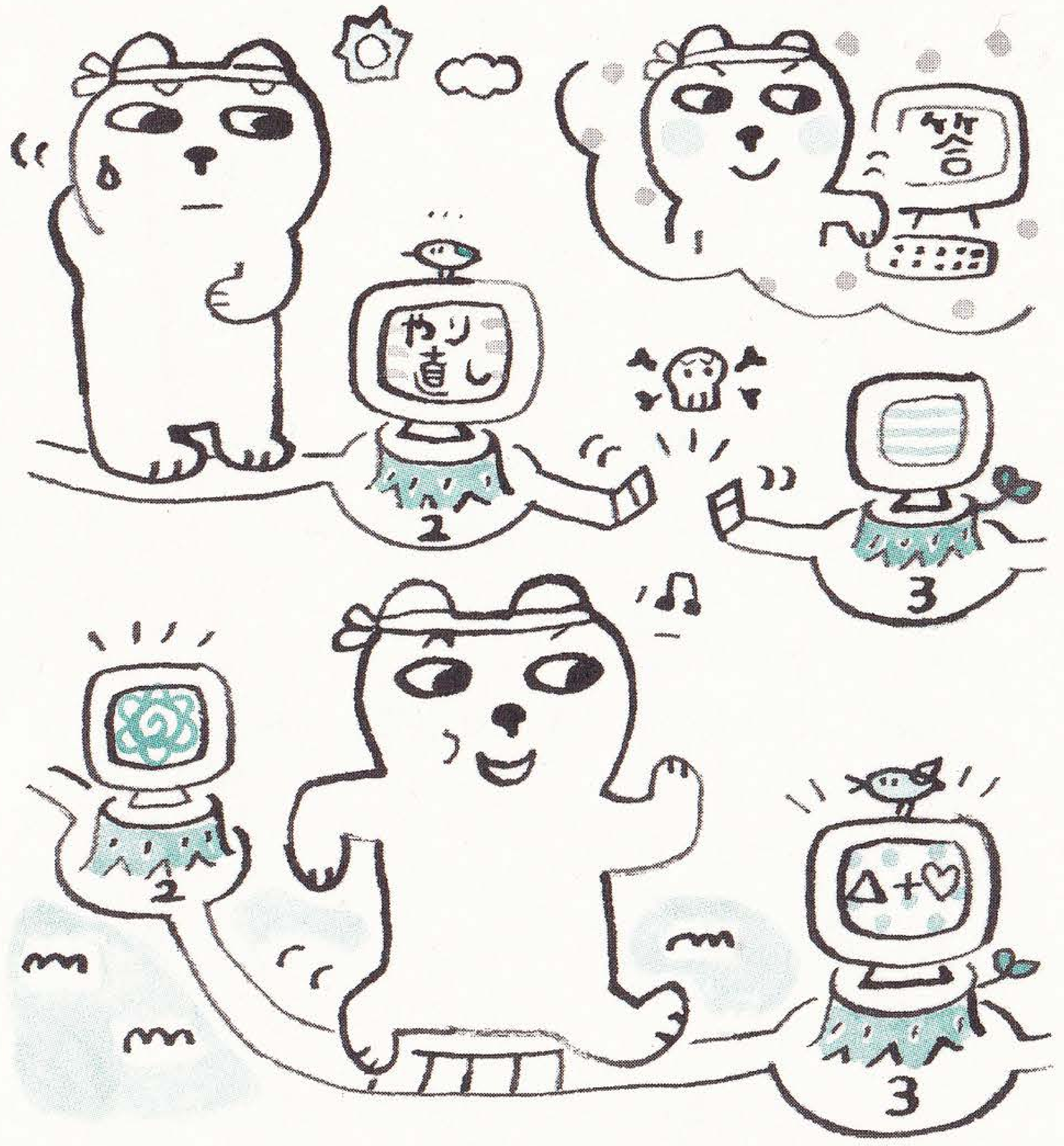
③くま　　　　表示された問題を解いて、答えをキー入力する。

④パソコン　　キー入力を受け付けて、入力されたデータが問題の計算結果と合うか確認する。

→合っていたら、次の問題を表示する。

→間違っていたら、同じ問題を表示する。

このように、③と④を繰り返せば、いいわけだ。





「ちょっと、待ってください。質問があります。このプログラムは、いつになったら終わるんですか？」



「そうだよね、ずっと特訓していたら、くま君おかしくなっちゃうね。」

では、とりあえず計算問題を10問やって、計算を「続けるか」「止めるか」を決めることにしよう。

次のように動けばいいかな。

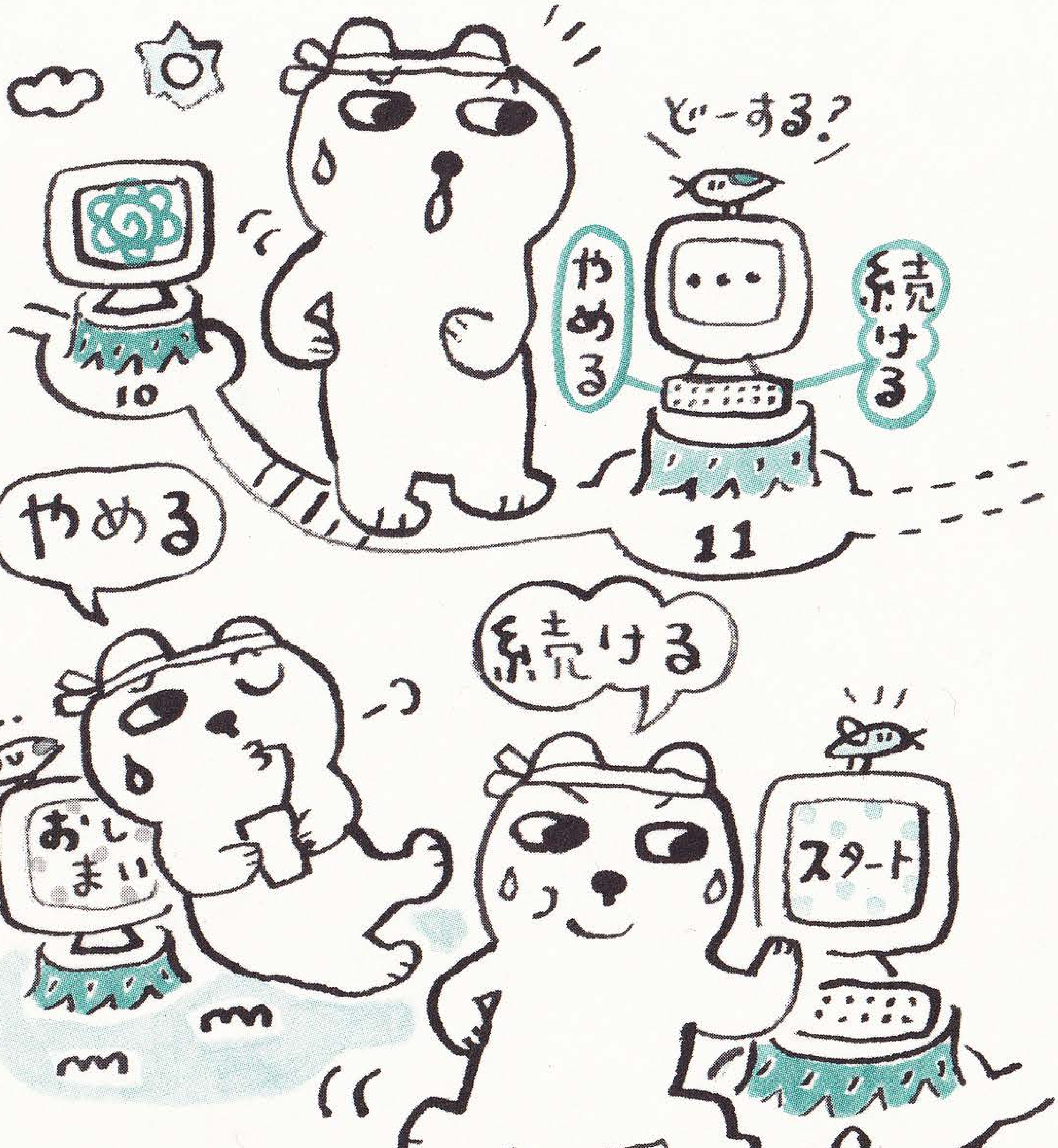
⑤パソコン　計算問題の表示回数が11回目だったら、問題は表示せずに、「計算を続ける」か「止めるか」を表示する（10回目以前なら、③〜⑤を繰り返す）。

⑥くま　「続ける」か「止める」かの選択をして、キー入力する。

⑦パソコン　→「続ける」のキーが入力されたら、表示回数を0に戻して、最初の問題を表示する。その後、③〜⑤を繰り返す。
→「止める」のキーが入力されたら、プログラムを終了する。



「これなら、わかりやすくて、安心できるね。」


10
どーする？
やめる
続ける
11
やめる
続ける
おしまい
スタート
0

さて、今までの内容をまとめてみよう。
これで、何を作ればいいのかがわかったね。

【プログラムの動き】

①くま　　「算数特訓プログラム」を動かす。

②パソコン　　プログラムの最初の問題を表示する。

③くま　　表示された問題を解いて、答えをキー入力する。

④パソコン　　キー入力を受け付けて、入力されたデータが問題の計算結果と合うか確認する。
→合っていたら、次の問題を表示する。
→間違っていたら、同じ問題を表示する。

⑤パソコン　　計算問題の表示回数が11回目だったら問題は表示せずに、「続ける」か「止めるか」を表示する（10回目以前なら、③〜⑤を繰り返す）。

⑥くま　　「続ける」か「止める」かを選択して、キー入力する。

⑦パソコン　　→「続ける」がキー入力されたら、表示回数を0に戻して、最初の問題を表示する。その後は③〜⑤を繰り返す。
→「止める」のキーが入力されたら、プログラムを終了する。

設計～どのように作るのか～

パソコンがどんな風に動けばいいのかが分かったので、
次にどのように動かすのか、その方法を考えていこう。
今まで勉強してきた変数や分岐条件、関数を使って分析の結果を実現する作戦を立ててみることにしよう。

設計では、検討したプログラムの動きの項目ごとに
実際にどんな風にプログラムを作っていけばいいか、
具体的に書き出していくんだ。



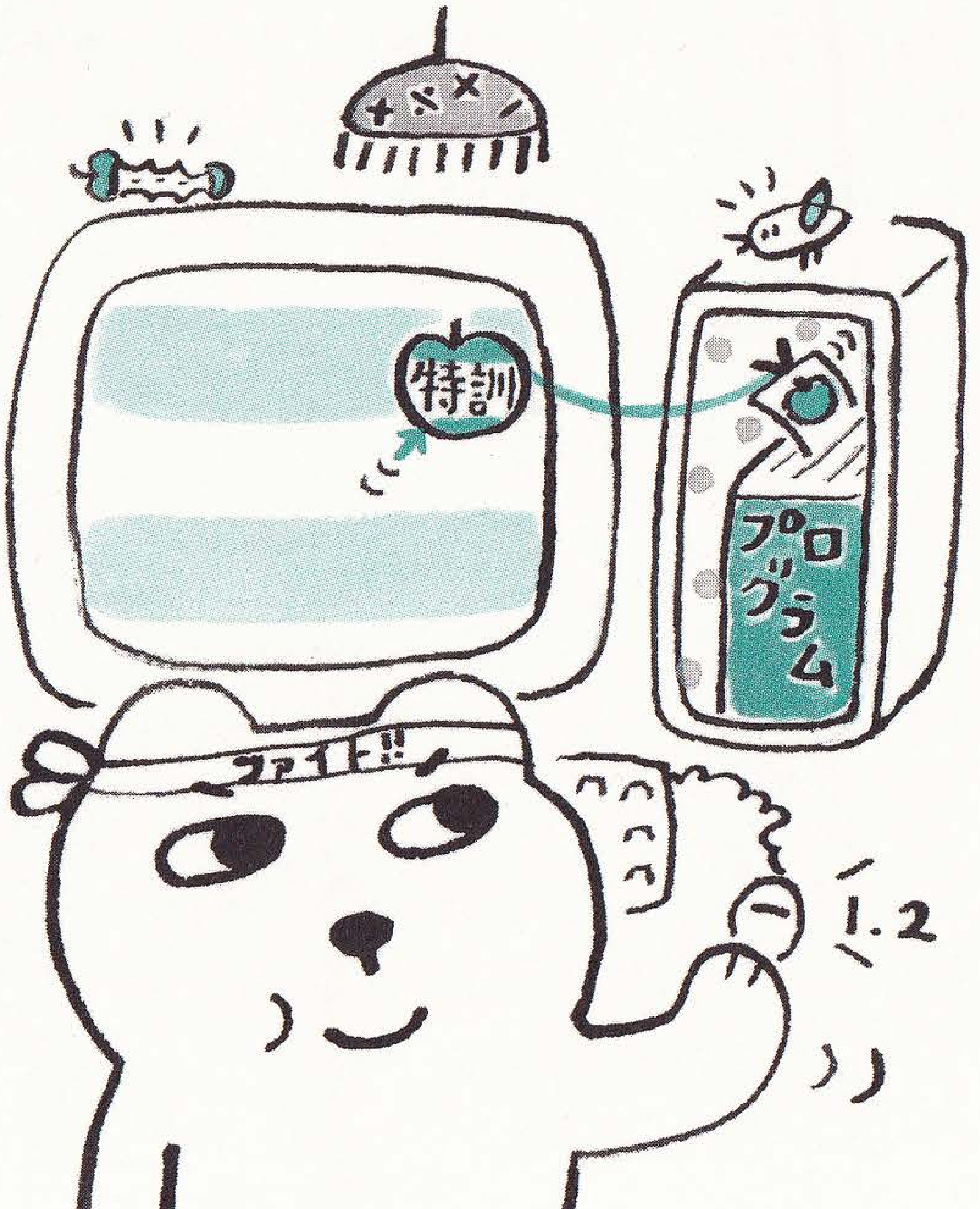
「はじめにプログラムで何をしていくのかを明らかにしたけれど、今度はどうやってコンピュータを動かすのか、その方法を確認していく、ということですね。」

その通りだ。
ひとつずつ確認していこう。

①くま 「算数特訓プログラム」を動かす

これは、くま君がパソコン上で[算数特訓プログラム]という実行ファイルを動かせばいいということだ。
このプログラムを動かすには[実行ファイル]を選択し、ダブルクリックすればいいんだ。

②パソコン プログラムを動かして、最初の問題を表示する。
　　　　　プログラムが起動してパソコンが実行する最初の処理だ。
　　　　　最初の「問題」に着目して、表示内容を明らかにしよう。



くま君、どんな問題を特訓したいのかな？



「とりあえず、基本的な足し算の練習からやってみたい。」

それじゃあ、足し算の問題を作成することにしよう。
この特訓では「2桁」足す「2桁」の計算を繰り返し行うようにする。

その時の画面の表示は、
「最初の数字」＋「2番目の数字」＝「結果」
と出るようにして、
表示方法は、printf()にすればいいね。
さて、この表示の中で「最初の数字」と「2番目の数字」を決めなければならないのだけれど、算数の特訓をするには、最初の数字と2番目の数字は、色んな組み合わせの数字であると便利だよね。

実は、プログラミングで色々な数字を組み合わせて使うことは結構多い。たとえば、パソコンでじゃんけんゲームを作ったり、宝くじの抽選番号を決めたりするには、色々な種類の数字がないと困るんだ。

そこで、C言語には、色々な数字（乱数）を発生するしかけとしてrand( )関数というのがある。このしかけを使って、最初の数字と2番目の数字を作り出すことができる（P223参照）。



「おー、すごい。この関数によって、色々な組み合わせの数字が出せるんですね。」

③くま 表示された問題を解いて、答えをキー入力する。

これは、くま君自身が表示された問題をやるところなんだ。
結果は1つなので、前に学習した通り（P141参照）、キー入力を取り込むしくみのscanf( )関数を使って、くま君がキー入力した数値を変数に記憶しよう。

④パソコン キー入力を受け付けて、入力されたデータが問題の計算結果と合うかを確認する。
→合っていたら、次の問題を表示する。
→間違っていたら、同じ問題を表示する。

これを少し分解して考えよう。
④-1 キー入力を受け付けて、入力されたデータが問題を計算した結果と合うか確認する（比較対象のデータを特定して判断する）。

キー入力は、変数に格納したから、その変数が計算した結果と合うかどうかを判断しなければならない。計算結果は別の変数に記憶して、この変数とくま君がキー入力した値の変数が等しければ、くま君の回答は正解となる。

④-2 合っていたら、次の問題を表示する（判断した結果でどちらかが選ばれる）

計算結果が正解だったら次の問題を表示する。
これは、最初の問題を表示する部分②に戻ればいい。

④-3 間違っていたら、同じ問題を表示する。

計算が間違っていたら、問題の式をそのまま表示しなければならない。
最初の数字と2番目の数字をそれぞれ変数にしておけば、
その変数をもう一度表示すればいい。

プログラムの流れをつかむために、このように分解すると、
条件分岐（P159参照）などが使えることがわかるし、流れがより明確になるね。

⑤パソコン　表示しようとしている計算問題の表示回数が11回目だったら問題は表示せずに、「続ける」か「止める」かを表示する（10回目以前なら、③～⑤を繰り返す）。

これも、④と同様に分解してみよう。

⑤-1 表示しようとしている計算問題の表示回数を確認する。

⑤-2 表示回数が11回目だったら、「続けるか」「止めるか」を表示する。

⑤-3 10回目以前なら③に戻る。

⑥くま 「続けるか」「止めるか」を選択して、キー入力する。

これは、③と同様にscanf( )関数を使って、キー入力を変数に入れる。

⑦パソコン 「続ける」がキー入力されたら、表示回数を0に戻して、最初の問題を表示する。その後は③〜⑤を繰り返す。「止める」がキー入力されたら、プログラムを終了する。

ここでも、手順④⑤と同様に分解してみよう。

⑦-1 「続ける」が入力されたら、表示回数を0に戻し、②の最初の問題を表示する。

⑦-2 「止める」が入力されたら、何もせずにプログラムを終了する。

さて、このプログラムでは、条件分岐が3つ必要になりそうだね。
分岐してプログラムの流れが変わるような時には、
フローチャートを作って確認するとよくわかる。

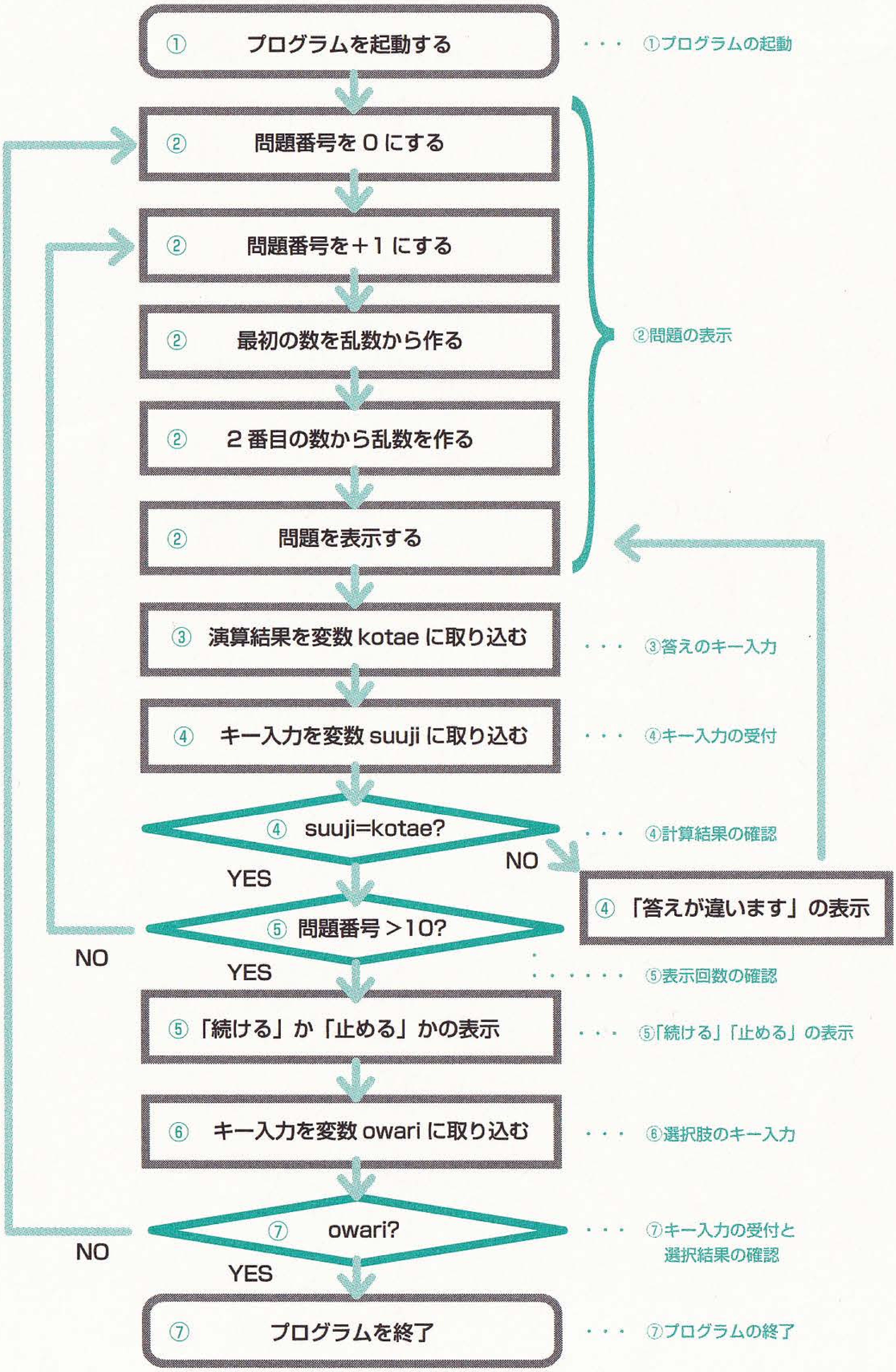


「ふろおちゃあと？」

フローチャートは、作業の手順を表す図のことだよ。
プログラム全体の流れを把握するのに便利なんだ。
この図を使うと、
プログラムの処理順序や条件分岐を表すことができるので、
どんな順番で処理を進めるのかがわかりやすくなるんだ。

実際にフローチャートを書いてみよう。
プログラムの始まりと終わりは、両端の丸い長四角形で書いて、処理は長四角で示す。そして条件分岐は、ひし形に書く。それぞれは矢印で結ぶと、処理の流れがわかるんだ。

フローチャートと今まで調べてきた①〜⑦の処理の流れを関連付けてみよう。



① プログラムを起動する
①プログラムの起動
② 問題番号を 0 にする
② 問題番号を＋1 にする
② 最初の数を乱数から作る
② 2 番目の数から乱数を作る
② 問題を表示する
②問題の表示
③ 演算結果を変数 kotae に取り込む
③答えのキー入力
④ キー入力を変数 suuji に取り込む
④キー入力の受付
④ suuji=kotae?
④計算結果の確認
NO
④ 「答えが違います」の表示
YES
⑤ 問題番号 >10?
NO
YES
⑤表示回数の確認
⑤「続ける」か「止める」かの表示
⑤「続ける」「止める」の表示
⑥ キー入力を変数 owari に取り込む
⑥選択肢のキー入力
⑦ owari?
⑦キー入力の受付と
選択結果の確認
NO
YES
⑦ プログラムを終了
⑦プログラムの終了

実装～ソースコードの作成～

では、実際に分析、設計で考えたものをプログラムにしてみよう。まずは、キー入力や表示が必要なので宣言をしていこう。

```
#include <stdio.h>        //標準関数を使用するための宣言
```

このプログラムでは、足し算の計算の時に、色々な種類の数を使う。乱数を発生させる標準関数も使用するので、もうひとつ標準関数を用意しよう。

```
#include <stdlib.h>       //標準関数を使用するための宣言
```

次に、main関数の中で使用する関数のプロトタイプ宣言をしておく。

```
int ransu(void);       //乱数を戻り値にする関数の宣言
```

いよいよ、main関数を書いてみよう。ここでのmain関数の戻り値はないので、void型で宣言しよう。

```
void main()
{
```

次に、この関数の中で使う変数の宣言をしていこう。

リスト5-1　変数の宣言

```
int saishonosuuji;          //最初の乱数の変数宣言
int nibanmenosuuji;         //2番目の乱数の変数宣言
```

```
int kotae;              //計算式の答えの変数宣言
int bango;              //問題の番号の変数宣言
int suuji;              //入力された答えの変数宣言
int owari;              //選択された動作を取得する変数の宣言
```

次に問題の繰り返しとその表示部分（問題の繰り返しと表示，P208参照）を作ろう。

リスト5-2　問題の繰り返し表示

```
START:                      //はじめから開始する時のラベル
  for( bango = 1 ; bango <= 10 ; bango++ ){
        //問題1～10まで繰り返す 問題の番号を1に初期化する
    saisyonosuuji = ransu();
    //乱数を返す関数を呼び出して最初の数字を取得する
  nibannmenosuuji = ransu();
  //乱数を返す関数を呼び出して二番目の数字を取得する
  kotae = saisyonosuuji + nibannmenosuuji;
  //取得した乱数で足算を実施し、kotae変数に入れる
    while(1){
      printf("問題");     //問題を表示する
      printf("%d : ", bango);
      printf("%d", saisyonosuuji);
      printf("+");
      printf("%d", nibannmenosuuji);
      printf("= ¥n");
      printf("計算結果を入力してください¥n");

      scanf("%d", &suuji);
      //入力された数を変数suujiに取得する
```

問題を繰り返して解く特訓をするために、戻るところを示すためのラベルをつけると次のようになるんだ。

```
START: //はじめから開始する時のラベル
```

そして問題の数を扱う処理は次のようになる。最初に問題の番号を1に初期化して、問題を10問繰り返すようにする。for文と条件式で書いてみよう。
ここでbango = 1 ; は、処理のはじまりを表わし、bango <= 10 ; は、変数bangoの値が10になるまで繰り返し、11になったら、このfor文を抜けて次の処理に進むということを表わしている。またbango++は、このfor文が繰り返される度に、変数bangoを1ずつ増やしていくことを示しているんだ。

```
    for( bango = 1 ; bango <= 10 ; bango++ ){
```

次に問題の作成をはじめる。乱数を返す関数を呼び出して最初の数字と2番目の数字を取得する。関数の戻り値は、2桁の乱数という前提で書いていこう。

```
saishonosuuji = ransu();
nibanmenosuuji = ransu();
```

足し算問題の足す数字と足される数字を作成できたので、ここで足し算の計算をしよう。取得した数の足し算結果を変数kotaeに入れる。

```
kotae = saishonosuuji + nibanmenosuuji;
```

次に、問題を表示するプログラムを書こう。問題の表示は、printf関数を使うけれど、その後にすぐキー入力を受け付けるので、while文を使って書いていく。

while文は｛｝の中に書いたプログラムを繰り返すように使うので、条件式にはいつでも条件が成立していることを示す“1”を直接入れておく。

```
while(1){
      printf("問題");
      printf("%d : ", bango);
      printf("%d", saishonosuuji);
      printf("+");
      printf("%d", nibanmenosuuji);
      printf("= ¥n");
      printf("計算結果を入力してください¥n");
```

次に、計算結果を入力することで、キー入力を受け付ける。

```
scanf("%d", &suuji); //入力された数を変数suujiに入れる
```

キー入力を受け付けたら、入力されたデータが問題の計算結果と合うかどうかの確認部分を作っていこう。合っていたら、次の問題を表示して、間違っていたら同じ問題を表示する（キー入力と入力結果の確認、P210参照）。

リスト5-3　問題の正解・不正解の確認

```
26          if(kotae == suuji){
27            break;
   // suuji＝kotaeなら正解なのでwhile文を抜けて次の問題
28          }
29          else{
30            printf("まちがい！　やりなおし！¥n");
                       //不正解の場合は同じ問題を繰り返す
31          }
32        }
33      }

              //特訓が終了し、プログラムの終了・再開の動作を選択
```

ここで、変数suuji＝変数kotae場合、正解なので、while文をbreak文で抜けることができ、13行目の次のfor文の処理ができる。for文の条件（10問正解）が成立するまで、次の問題の計算や表示、入力、確認を繰り返して行うんだ。

```
26          if(kotae == suuji){
27            break;
   // suuji＝kotaeなら正解なのでwhile文を抜けて次の問題
28          }
```



「じゃあ、問題を解いているときに間違ってしまったときは、どうなるの？」

正解以外のとき、つまり不正解の場合は、if文の条件が成立しないので、break文の処理ができず、while文を抜けることができない。正解になるまで、「間違いである」ことを表示させることにしよう。



「まちがい！とか、やりなおし！とかだね。」

そうだね、そうしよう。間違えた場合というのは、さっきのif文の条件が成立しないということなので、elseの括弧の中に「まちがい！やりなおし！」と表示し、同じ問題を繰り返すようにするんだ。

```
        else{
          printf("まちがい！ やりなおし！¥n");
                //不正解の場合は同じ問題を繰り返す
        }
      }
    }
```

ここまでで、10問を繰り返すプログラムができた。

残りは、10問の問題に正解した後のプログラムを書いていくよ（続行・終了の選択，P212参照）。10問の特訓が終了して「続ける」または「終わる」のかを選ぶ場面だ。やはり表示の後にキー入力を確認しないといけないので、また、while文で作ってみよう。

リスト5-4　問題の続行・終了

```
while(1){
  printf("よくできました！ 10問正解です¥n");
  printf("はじめからしますか？終了しますか？¥n");
  printf("1：はじめから¥n");
  printf("2：終了¥n");

 scanf("%d", &owari);
 //選択された番号を変数owariに取得する
    if( owari == 1 ){
      goto START;
//owariが1の場合はSTARTラベルの位置にもどって特訓の再開
    }
    else if( owari == 2 ){
      break;  //owariが2の場合はプログラムを終了する
    }
  }
}
```


ここでは10問正解後に表示する「はじめからしますか？」と「終了しますか？」の案内画面を書いていく。

```
while(1){
  printf("よくできました！ 10問正解です¥n");
  printf("はじめからしますか？終了しますか？¥n");
  printf("1：はじめから¥n");
  printf("2：終了¥n");
```

選択肢を表示したら、「1」か「2」のキー入力を受け付ける。

```
    scanf("%d", &owari);
    //選択された番号を変数owariに取得する
```

変数owariが“1”の場合は、STARTラベルの位置から特訓を再開するので、「START」というラベルの行に戻る。goto文を使うと、簡単にラベルの行に戻ることができるので、これを使ってみよう。

```
        if( owari == 1 ){
          goto START;
        }
```

変数owariが“2”の場合は、特訓を終了するのでプログラムを終了すればいいね。この時、else ifで記述することにしよう。

```
        else if( owari == 2 ){
          break;  //owariが2の場合はプログラムを終了する
        }
      }
    }
```

ここまででmain関数は終了だ。

最後に、続けて足される数字と足す数字を乱数で作る関数をプログラミングしていこう（乱数の作成，P209参照）。

main関数で呼び出す最初の数字と2番目の数字を戻り値にするように関数を作っていくんだ。

戻り値は、int型に決め、関数名は、「ransu」とする。関数名の前のデータ型が、戻り値のデータ型を示すんだったよね。また、この関数

に渡す引数はないので、関数名の後の（ ）の中には、voidと書いておこう。

リスト5-5　乱数で数字を作る

```
int ransu(void)
{
    int suuji;//標準関数から取得した乱数を入れる変数の宣言
    int max; //桁数に応じた乱数の最大値の変数宣言
    int min; //桁数に応じた乱数の最小値の変数宣言

    max = 99;                //最大値は99
    min = 10;                //最小値は10

    while(1){
      suuji = rand();   //標準関数から乱数を取得する
      if(( suuji <= max ) && ( suuji >= min )){
        break;
//取得した乱数がmax以下、min以上であればwhile文を終了する
      }
    }
    return(suuji); //取得した乱数を戻り値として返す
  }
```

最初は、この関数内で使用する変数の定義からだ。

```
  int suuji;  //標準関数から取得した乱数を入れる変数の宣言
  int max;          //桁数に応じた乱数の最大値の変数宣言
  int min;          //桁数に応じた乱数の最小値の変数宣言
```

次に定数項を設定する。

ここで定数項として、maxとminを設定しておく。
くま君の特訓は2桁の足し算なので、関数ransuの戻り値は、10進数で2桁のint型データにする。
決定した値が10以上、99以下なら条件に合うので、結果が最小値以上、最大値以下にならなかったら、もう一度計算し直すようにする。

```
        max = 99;               //最大値は99
        min = 10;               //最小値は10
```

引き続きwhile文を用いて、
2桁の乱数を特定する部分まで作ってみよう。

```
        while(1){
```

次の標準関数randは、int型の戻り値を戻してくれる関数なんだ。
この戻り値を変数suujiに入れるようにしよう。

```
        suuji = rand();
```

変数suujiがmax以下、min以上であればbreak文を抜けてwhile文を終了する。このif文の条件が成立しない場合は、while文の最初のsuuji = rand( ); から繰り返すんだ。

```
        if(( suuji <= max ) && ( suuji >= min )){
          break;
    //取得した乱数がmax以下、min以上であればwhile文を終了する
        }
      }
```

数値が決まったら、61行目の戻り値suujiを14行目や15行目saisy-onosuuji , nibanmenosuujiに渡して、呼び出したmain関数に戻る。

```
61      return(suuji); //取得した乱数を戻り値として返す
62    }
```

リスト5-6　算数特訓プログラム

```
01 #include <stdio.h>     //標準関数を使用するための宣言
02 #include <stdlib.h>    //標準関数を使用するための宣言
03 int ransu(void);       //乱数を返す関数の宣言

04 void main()
05 {
06 int saisyonosuuji;    //最初の乱数の変数宣言
07 int nibannmenosuuji; //二番目の乱数の変数宣言
08 int kotae;           //計算式の答えの変数宣言
09 int bango;           //問題の番号の変数宣言
10 int suuji;           //入力された答えの変数宣言
11 int owari;           //選択された動作を取得する変数の宣言

12 START:               //はじめから開始する時のラベル
13   for( bango = 1 ; bango <= 10 ; bango++ ){
            //問題1～10まで繰り返す 問題の番号を1に初期化する
14     saisyonosuuji = ransu();
            //乱数を返す関数を呼び出して最初の数字を取得する
15    nibannmenosuuji = ransu();
            //乱数を返す関数を呼び出して二番目の数字を取得する
16      kotae = saisyonosuuji + nibannmenosuuji;
            //取得した乱数で足算を実施し、kotae変数に入れる
```

```
        while(1){
            printf("問題");     //問題を表示する
            printf("%d : ", bango);
            printf("%d", saisyonosuuji);
            printf("+");
            printf("%d", nibannmenosuuji);
            printf("= ¥n");
            printf("計算結果を入力してください¥n");

            scanf("%d", &suuji);//入力された数を変数
    suujiに取得する

            if(kotae == suuji){
                break;
        // suuji＝kotaeなら正解なのでwhile文を抜けて次の問題
            }
            else{
                printf("まちがい！ やりなおし！¥n");
                            //不正解の場合は同じ問題を繰り返す
            }
        }
    }
        //特訓が終了し、プログラムの終了・再開の動作を選択
while(1){
    printf("よくできました！ 10問正解です¥n");
    printf("はじめからしますか？終了しますか？¥n");
    printf("1：はじめから¥n");
    printf("2：終了¥n");
```

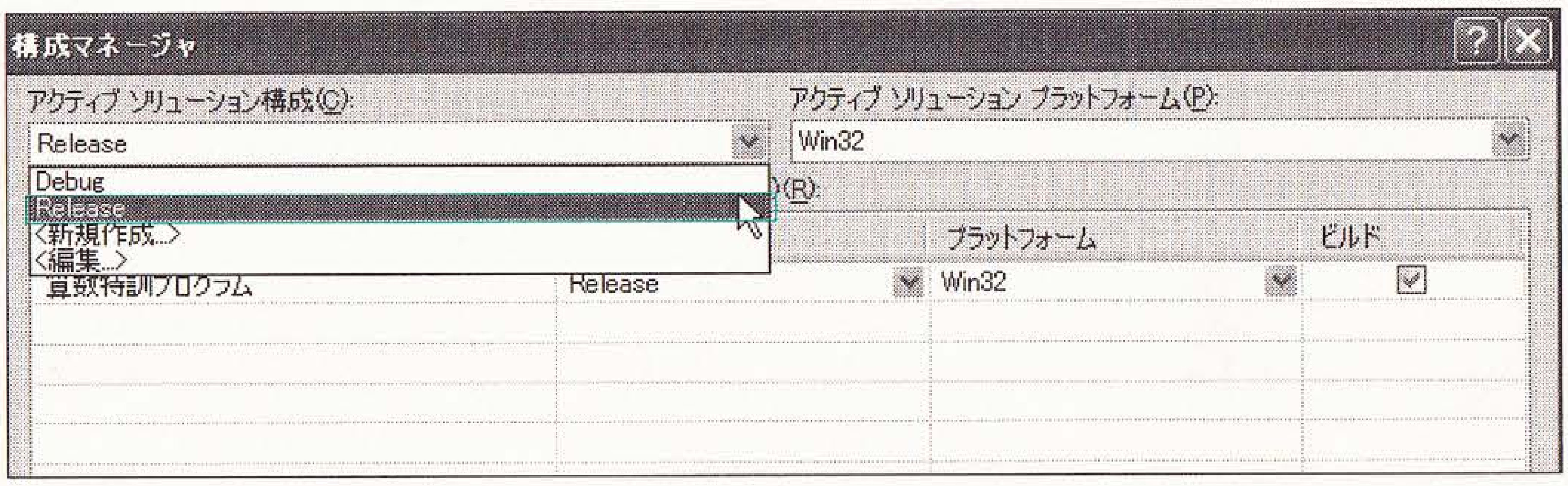
```
  scanf("%d", &owari);
  //選択された番号を変数owariに取得する
   if( owari == 1 ){
    goto START;
    //owariが1の場合はSTARTラベルの位置にもどって特訓
の再開
   }
   else if( owari == 2 ){
    break;   //owariが2の場合はプログラムを終了する
   }
  }
}
int ransu(void)
{
  int suuji;//標準関数から取得した乱数を入れる変数の宣言
  int max; //桁数に応じた乱数の最大値の変数宣言
  int min; //桁数に応じた乱数の最小値の変数宣言

  max = 99;            //最大値は99
  min = 10;            //最小値は10

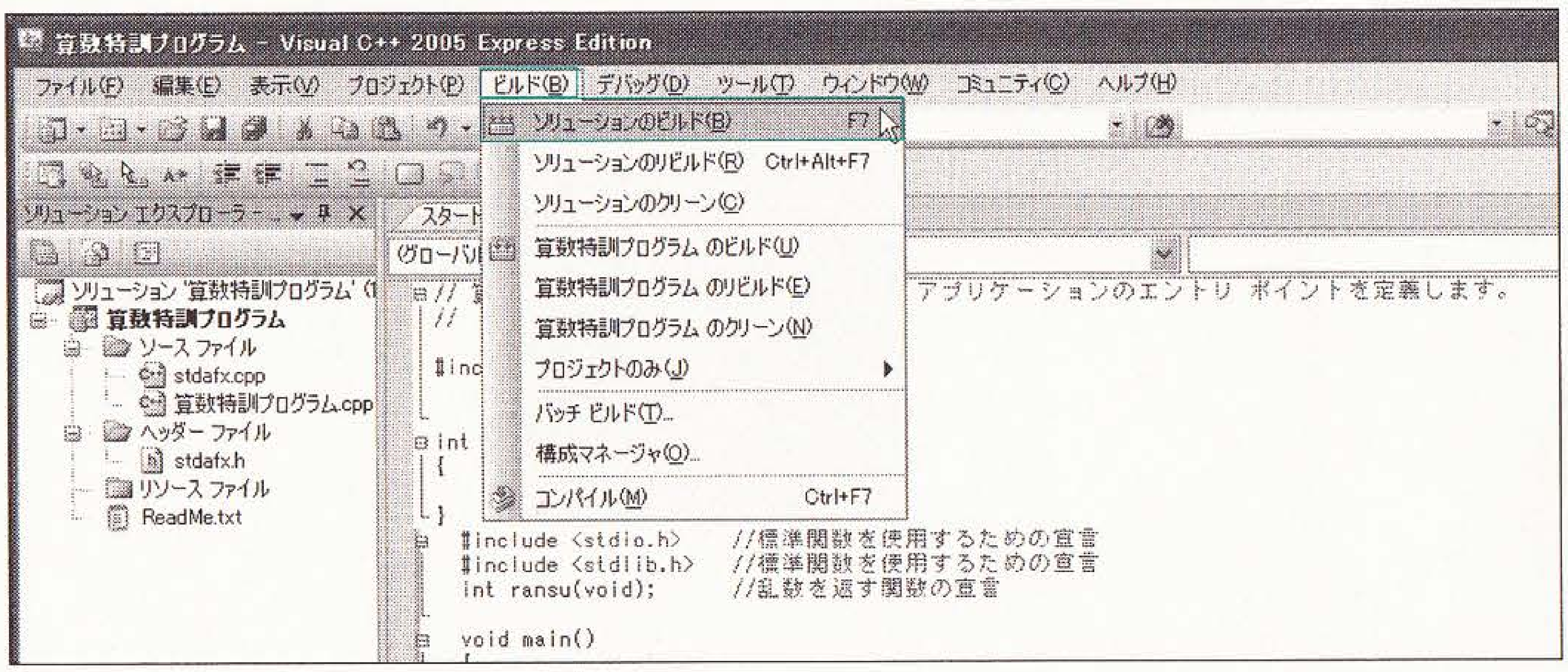
  while(1){
   suuji = rand();  //標準関数から乱数を取得する
   if(( suuji <= max ) && ( suuji >= min )){
    break;
 //取得した乱数がmax以下、min以上であればwhile文を終了する
   }
  }
  return(suuji); //取得した乱数を戻り値として返す
 }
```

これでプログラムの記述が終わったので、コンパイルしてみよう。

1. メニューから[ビルド]-[構成マネージャ]を開き[アクティブソリューション構成]を「Release」にして閉じる。次にメニューから[ビルド]-[ソリューションのビルド]を選択する。



● [構成マネージャ] を [Release] に

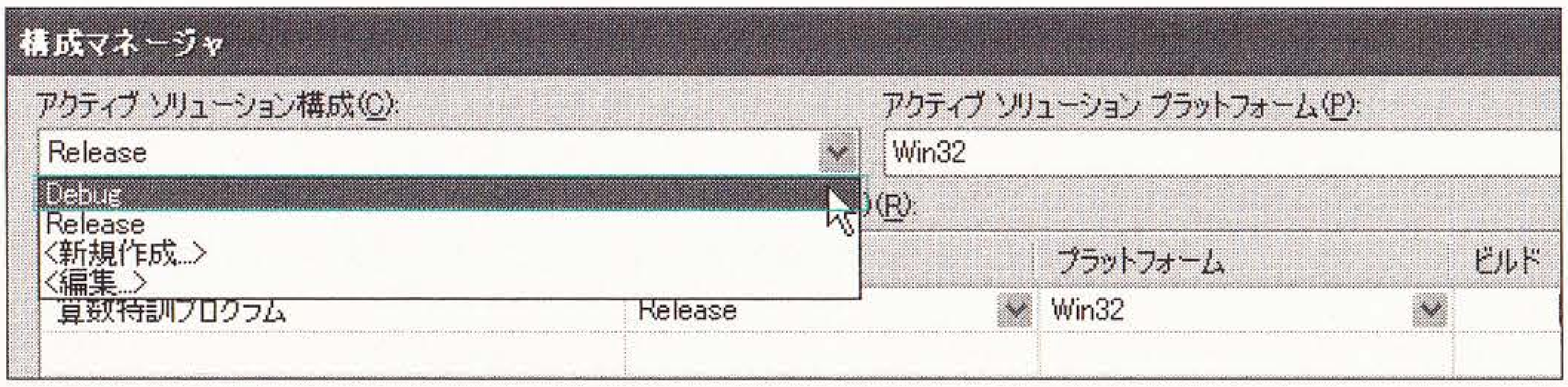


● コンパイルからリンクの実行

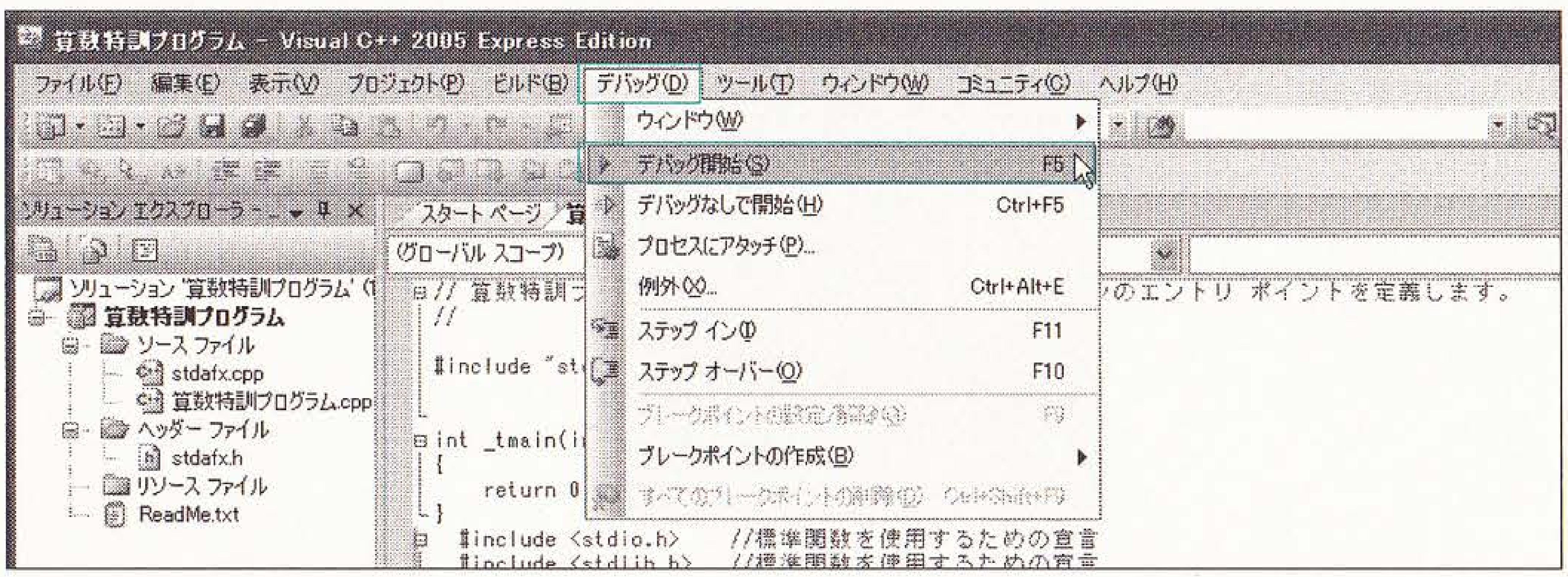
コンパイルが終了すると、実行ファイルができあがる。¥releaseフォルダの中に算数特訓プログラム.exeができるね。次にデバッグを開始してプログラムを実行してみよう（ここではプログラムの実行とデバッグを同時に行う）。

2. メニューから[ビルド]-[構成マネージャ]を開き[アクティブソリューション構成]を「Debug」にし、閉じる。次にメニューから[ビルド]-[ソリューションのビルド]を選択して、デバッグの準備をする。

3. F5キーを押し、デバッグをする（メニューから[デバッグ]-[デバッグ開始]をクリック）。



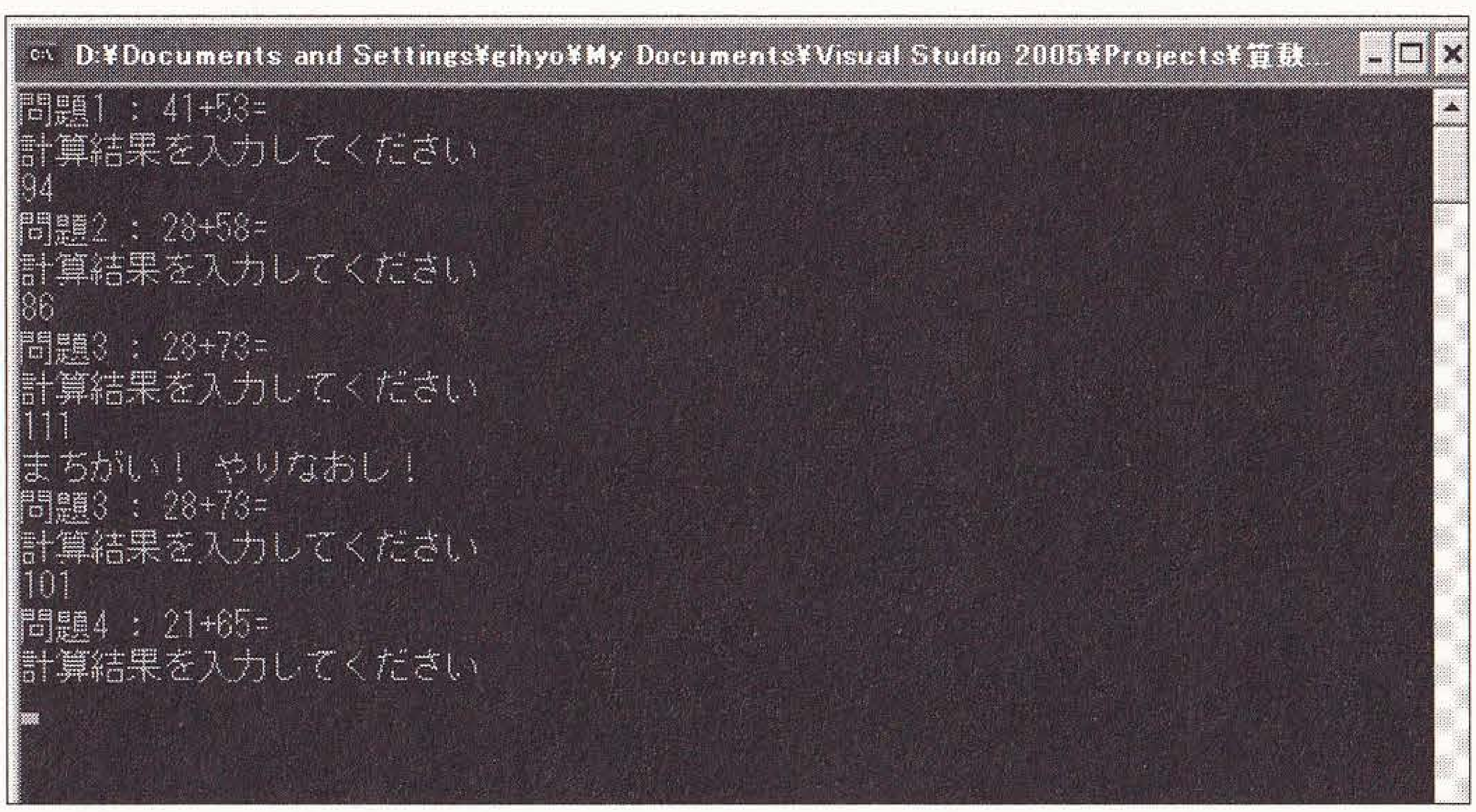
● ［構成マネージャ］を［Debug］に



● プログラムの実行



「プログラミングって、大変だけど、なんだか面白いですね。」



● プログラムの完成

テスト～動くか試してみる～

デバッグが終了したので（P 228参照）、ここでは実際にプログラムを動かしてみて、くま君の算数特訓に本当に役立つか試してみよう。
その際に、どんな風に動かそうとしていたのかを見直して、**テスト**項目を作ってみよう。
テスト項目は、設計の時に作った**フローチャート**の流れをすべて通るように考える。さらに、フローチャートにはない動かし方も考えて、実行しておくといいね。
テストをするときには、普通に使ったときに正しく動くことと、答えを間違ったときに、誤った動きをしないことを確認するんだ。
普通に使えるというのは、くま君がプログラムを起動してから、操作を間違えることなく、特訓できることをいう。
くま君とパソコンの間では、くま君のキー入力だけでやりとりをするので、くま君がキー入力する値に着目してテストすればいいんだ。


「ぼくのキー入力ですか？」

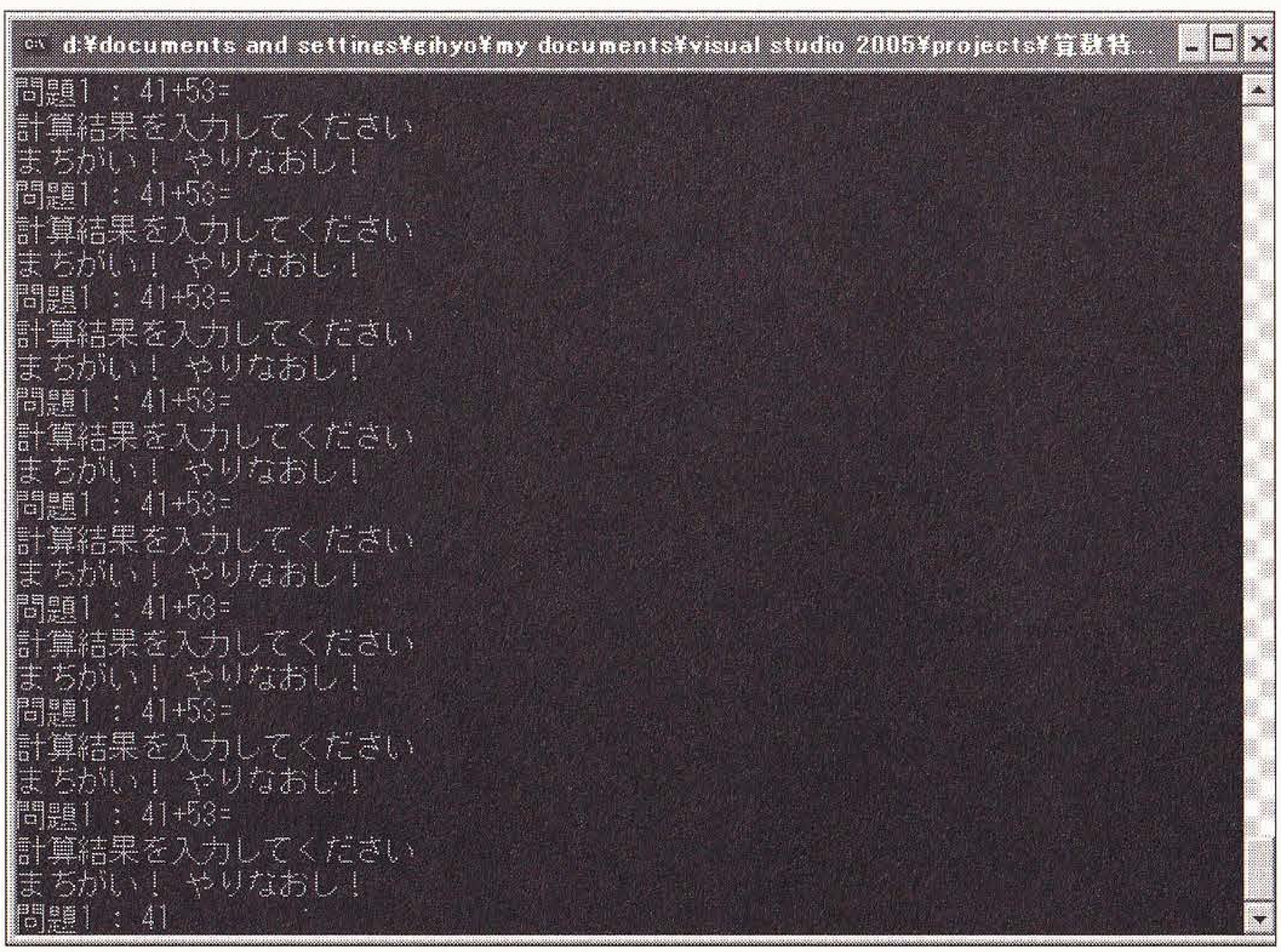
そうだね、まずは、まじめに特訓しようとすることを考えよう。つまり、パソコンの表示にしたがって、数字のキーを入力することを前提にする。
設計の時に作ったフローチャートの判断（**条件分岐**）がうまく動いているか、実際に動かせばいいね。

問題は、操作を間違えたときの確認なんだよ。
たとえば、問題の答えのキー入力は、数字で入れなければいけないんだけど、間違って、アルファベットを入れるとどうなるかな？
この場合は、プログラムで定義したデータ型と異なるため、scanf関数が暴走してしまうんだ。


「ぼうそう？」

実際にやってみればわかるよ。


「わー、止まらないよー。」



操作間違い

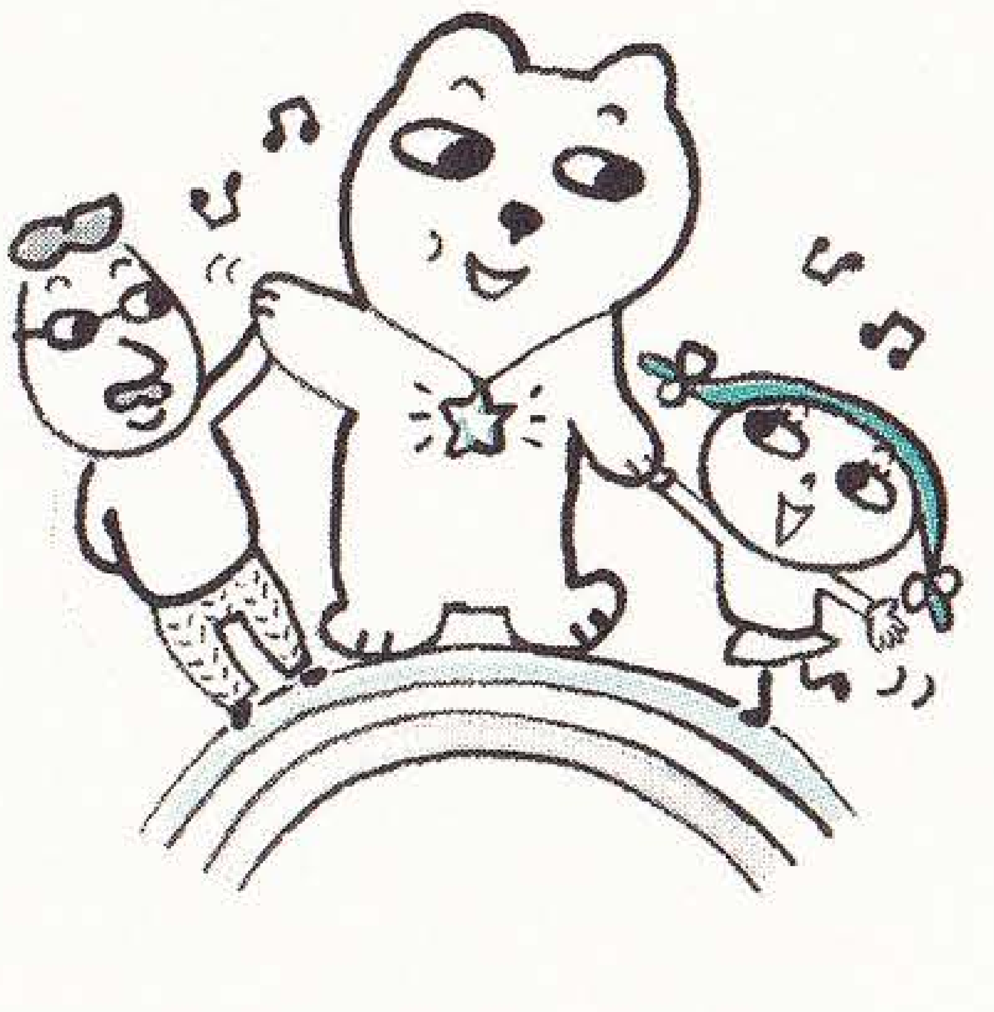
作成したプログラムが暴走したときは、DOS窓を閉じれば強制終了するようにできているので、落ち着いて止めよう。

このように、プログラムの作り方によっては、操作を間違えたときに、パソコンが**誤って**動くことがある。
そのため、できあがったプログラムは、十分にテストして、どのように動くかということを確認しておいたほうがいいね。



「なるほど。自分でテストすれば、間違ったときに、慌てないですみますね。」

第6章
まとめ



完成したプログラムが動くところまで確認できて、喜ぶ絵美とくま。
でも、くまにとって一番つらいのは、これからはじまるパソコンによる訓練だった。
くまは、算数特訓プログラムの計算の答えが合わないと、自分の能力を疑うでもなく、プログラムのどこかに不具合（間違い）があるのではないかと疑心暗鬼になるのであった･･･。

くま「何で答えが合わないんだろう。僕の答えは合っているはずなのに。このプログラムどこかおかしいんじゃないかな･･･。」

くまは、何度も何度も同じところでつまずいている。
相当に疲労が溜まっている、くまの後ろ姿であった…。


しずかに!!

6-1 コンピュータの動き

Power On

プログラムも完成したし、あとは、くま君が特訓を繰り返すだけだな。
くま君、まさかコンピュータと格闘みたいなことになっていないだろうな。


「ちょっと心配だね…。」


**「先生、先生…。大変です。答えが合わないんです。
きっと、パソコンの答えが間違っていると思うのですが…。」**

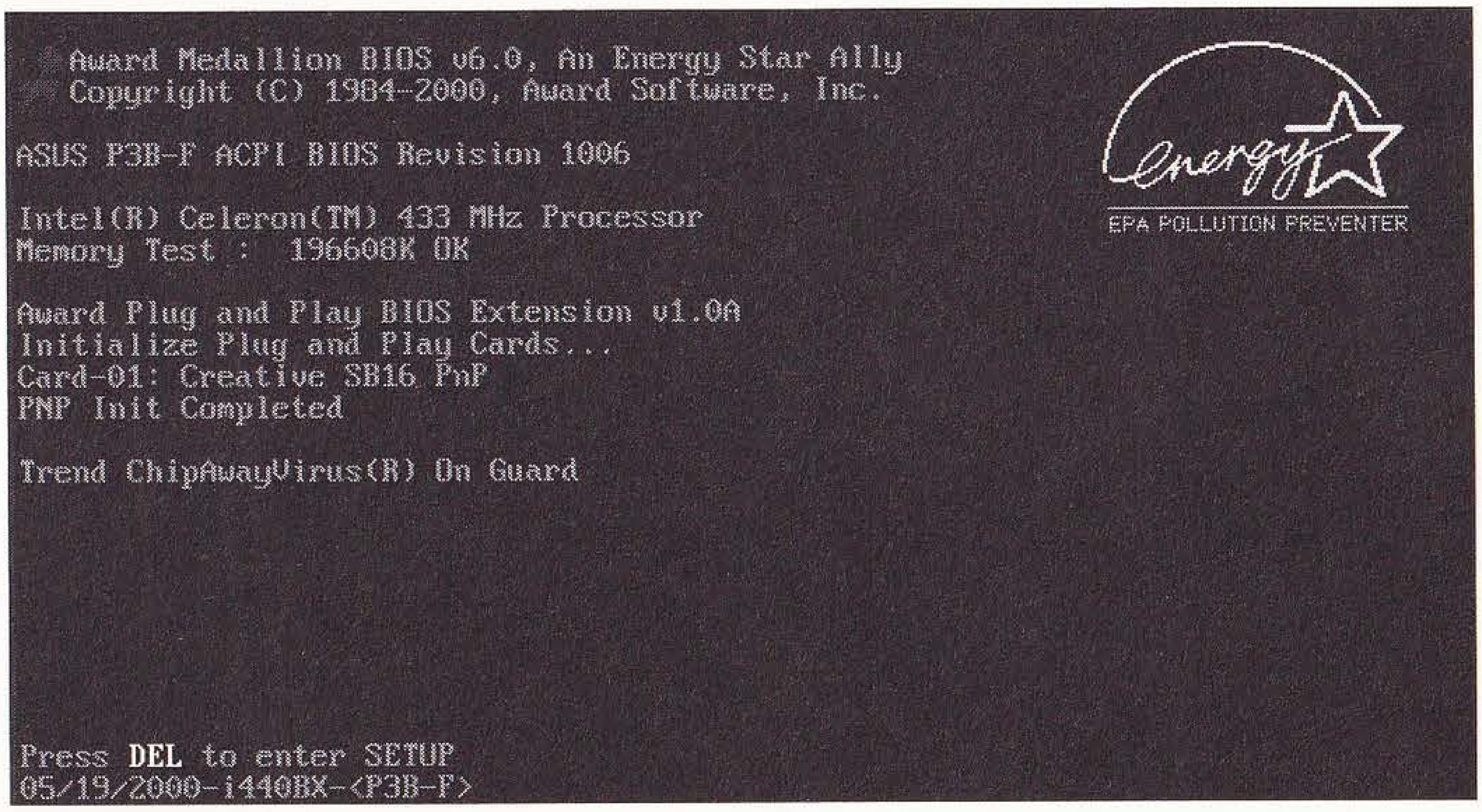
やっぱり、そんなことをいいだすのではないかと思っていたところだよ。
残念ながら、プログラム自体に間違いはなさそうだ。
くま君、今までの勉強を総ざらいしてみよう。
プログラムの動きがしっかり理解できれば、プログラムを信頼できるからね。


「ハイ…。」

まずは、
パソコンの電源を入れる（ONにする）ところから復習しよう。
パソコンの電源を入れると、ランプがついてカラカラと音がする。

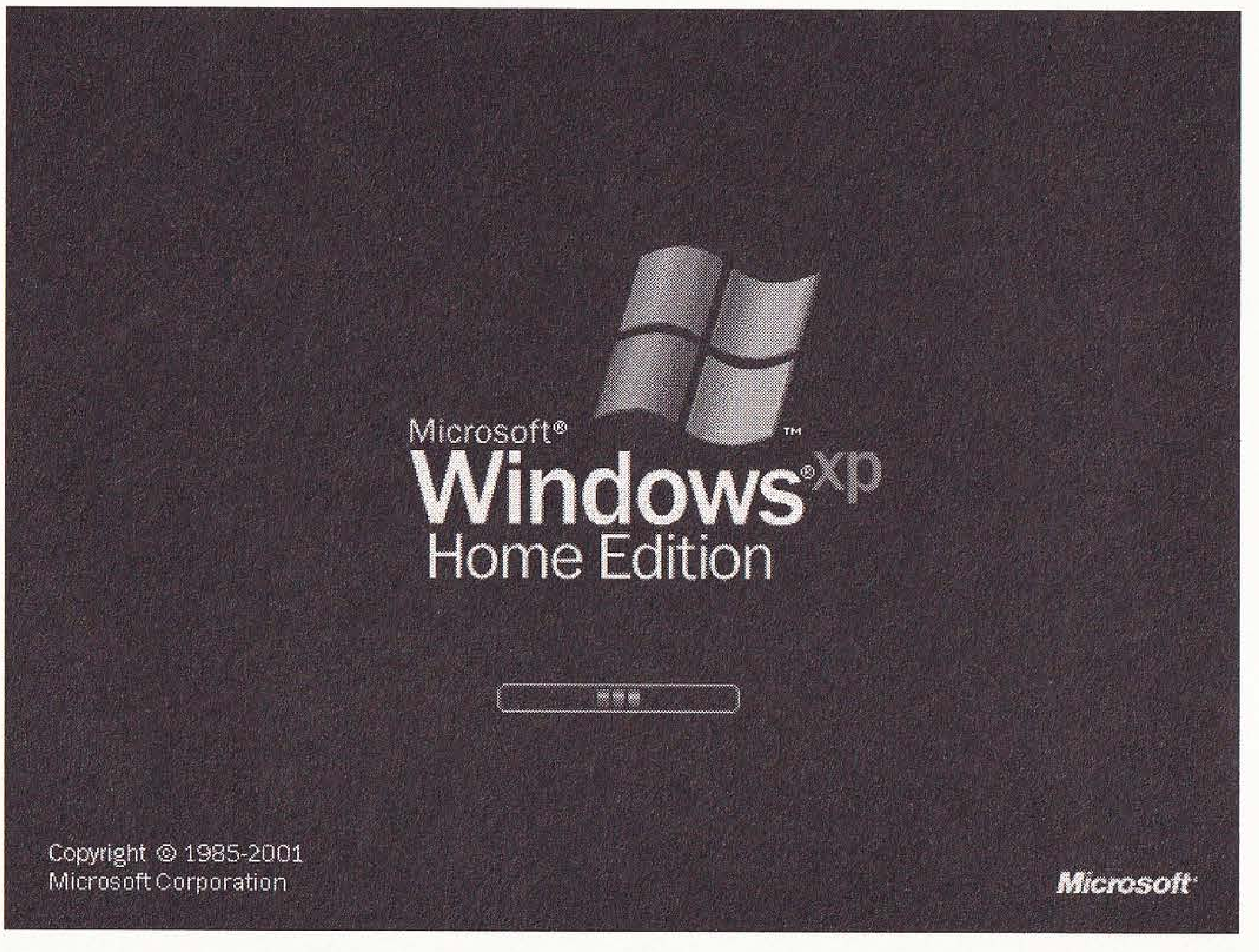
電源が入ると、ブートプログラム（パソコンに電源を入れたとき一番初めに実行される小さなプログラムのこと）が動いて、ハードウェアを初期化する。

初期化とは、パソコンが使えるように、パソコンのハードウェアやディスプレイ、キーボードなどの周辺装置を設定することをいうんだ。



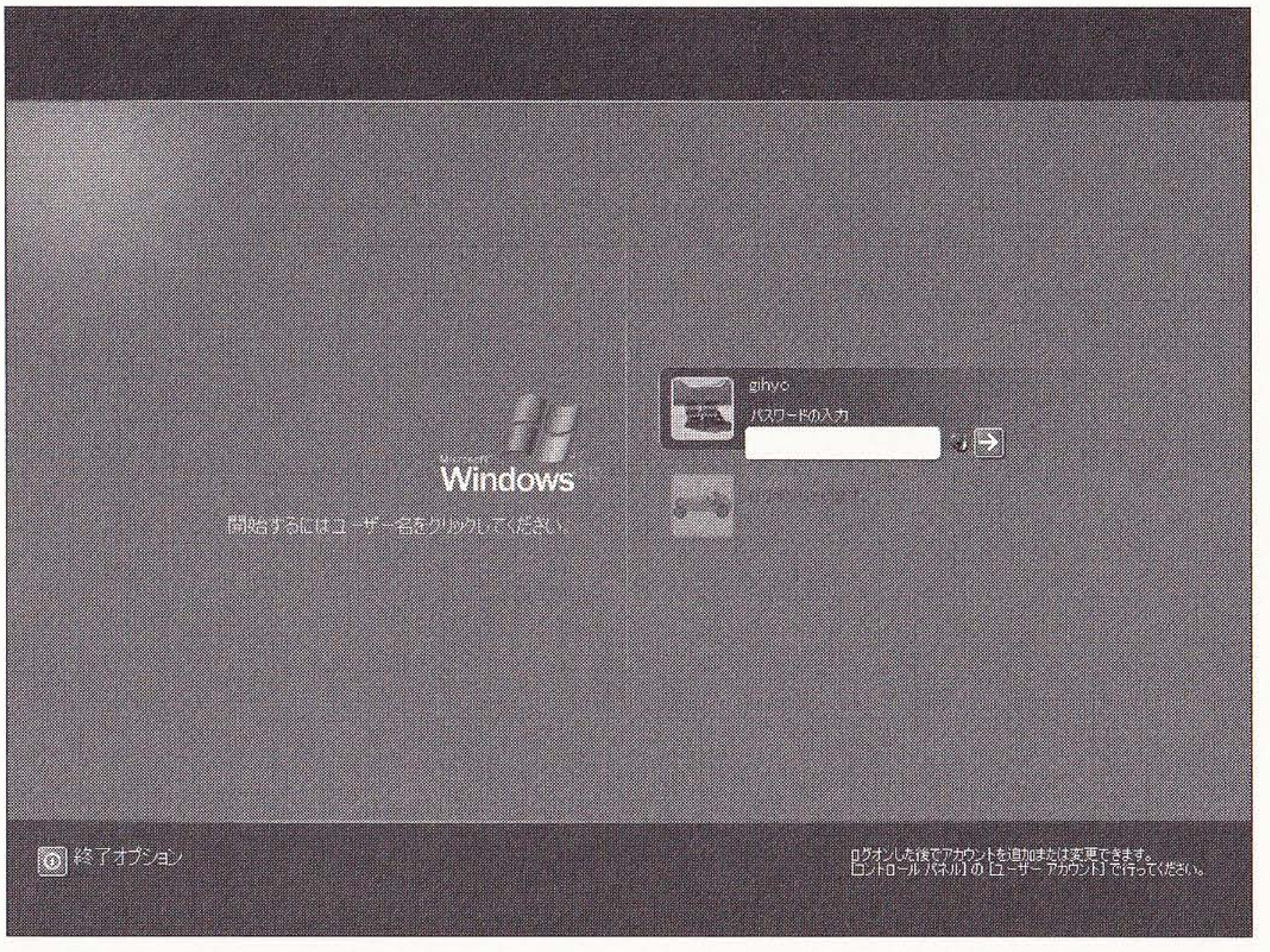
初期化

初期化が終わると、ハードディスクにアクセスし、いよいよOS（基本ソフト）が動き始めて、プログラムを動かすための準備を始める。準備が始まると、標準出力には、準備中を意味するロゴやマークが表示されるんだ。



準備中〜ロゴやマークの表示〜

その後、パソコンの準備ができたら**ログイン画面**を表示する。
パソコンを使う側にとってログインするのが最初の仕事だね。

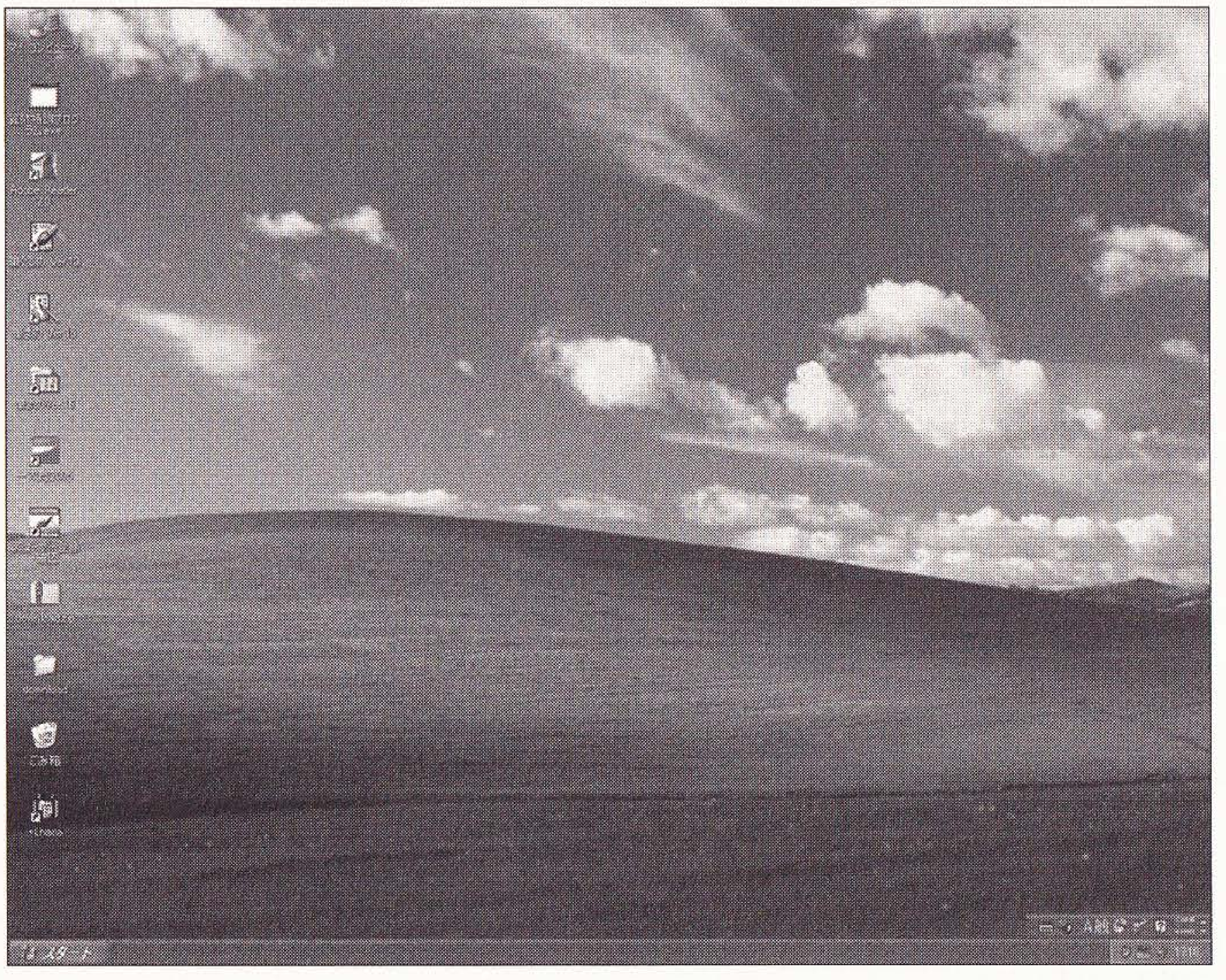


● ログイン

ログインが行われると、**デスクトップ**が表示される。
そしていよいよ、プログラムを動かす準備が万端になるんだ。
ログインして何もしなければ、そのままの状態を維持する。
これを**待機状態**というんだ。


「パソコンのたいき状態って一体何…？」

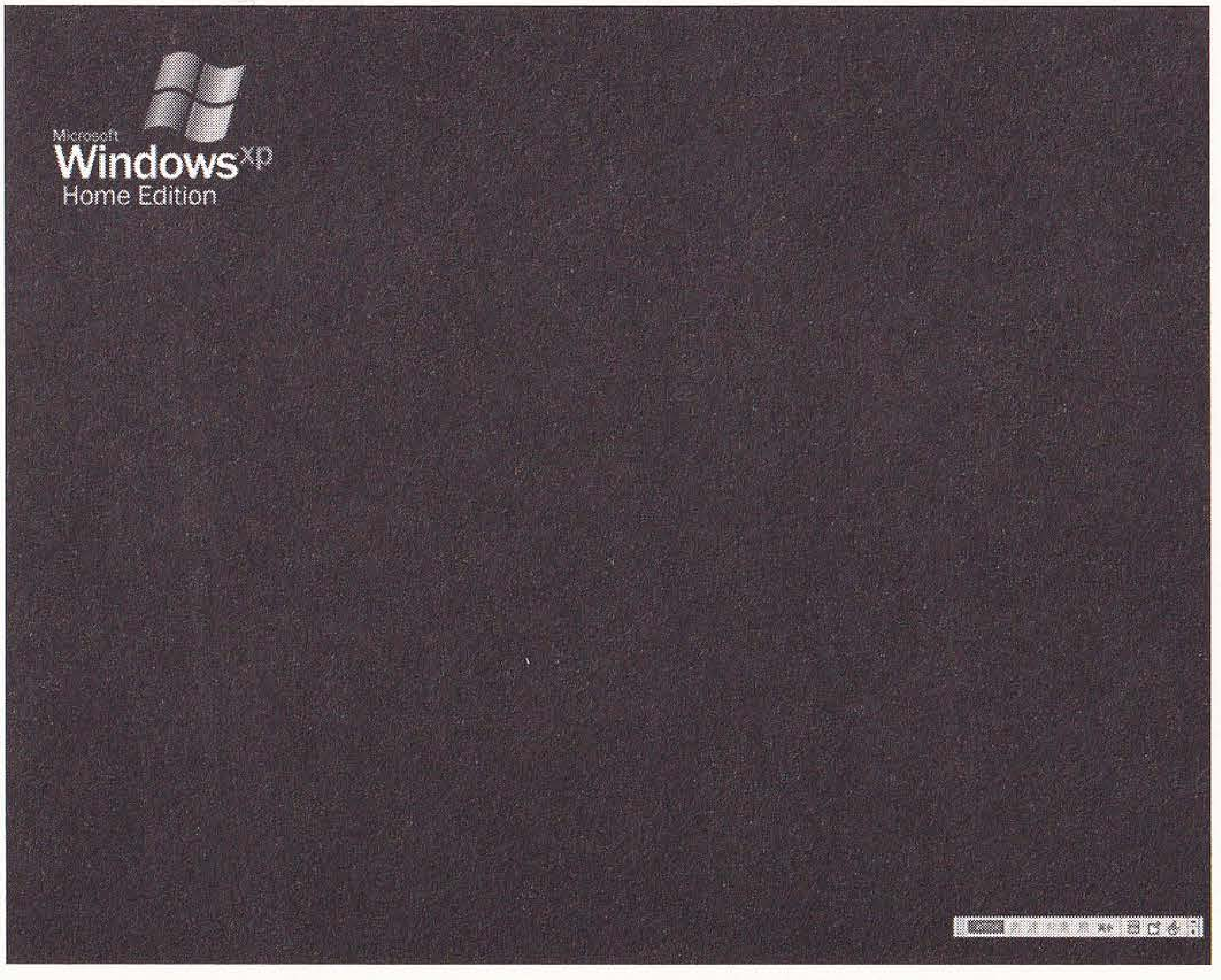
プログラムを動かす準備が完了して、コンピュータがプログラムを動かすための指示を待っている状態のことをいうんだ。
つまり、デスクトップが表示されたままで、プログラムが起動されていない状態のことをいう。



待機状態

「でも、このままにしておくと、画面が変わってしまうよ。」

これは、**スクリーンセーバー**の働きだね。
ログインしたときにタイマーを仕掛けていて、そのタイマーがタイムアウトしたときに、スクリーンセーバーのソフトが動くようになっているんだ。
ちなみに、この設定がされていると、もう一度ログインが必要な場合もある。



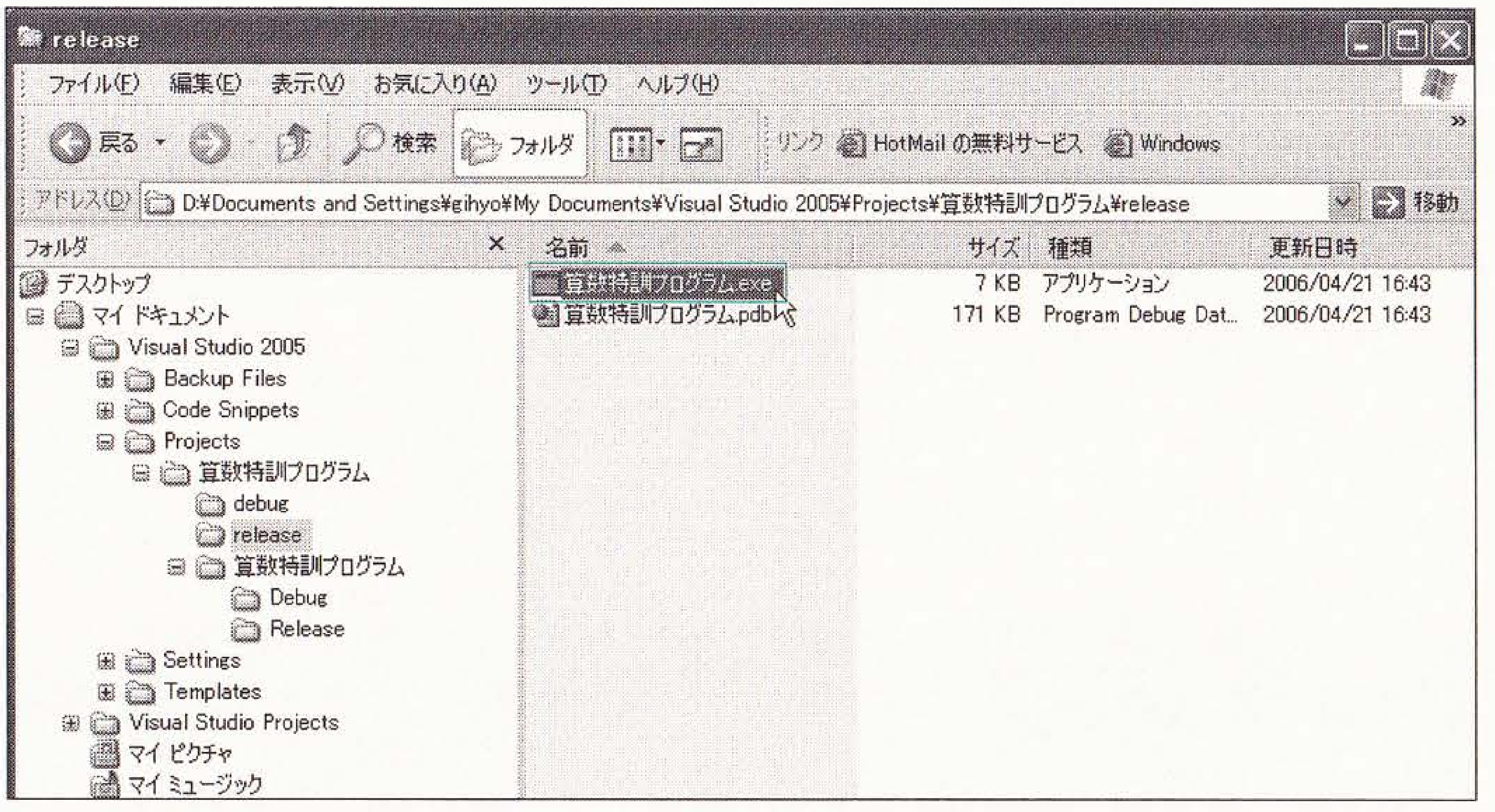
スクリーンセーバー

6-2 プログラムの動き

プログラムの起動

ログインが終わったら、早速、くま君の特訓プログラムを動かしてみよう。プログラムとパソコンの動きが整理できれば、パソコンがプログラムの通りに正しく問題を出して、くま君の回答結果を正しく判定していることがわかるよ。

ログインすると、まずデスクトップが現れるよね。
ここで、くま君がいつでも算数の特訓ができるように、デスクトップ上に**実行ファイル**を置いておこう。
実行ファイルは、プログラムを作ったフォルダのreleaseフォルダ（D:¥Documents and Settings¥gihyo¥My Documents¥Visual Studio 2005¥Projects¥算数特訓プログラム¥release）の中にあるので、このフォルダをダブルクリックして、開いてみよう。



実行ファイル



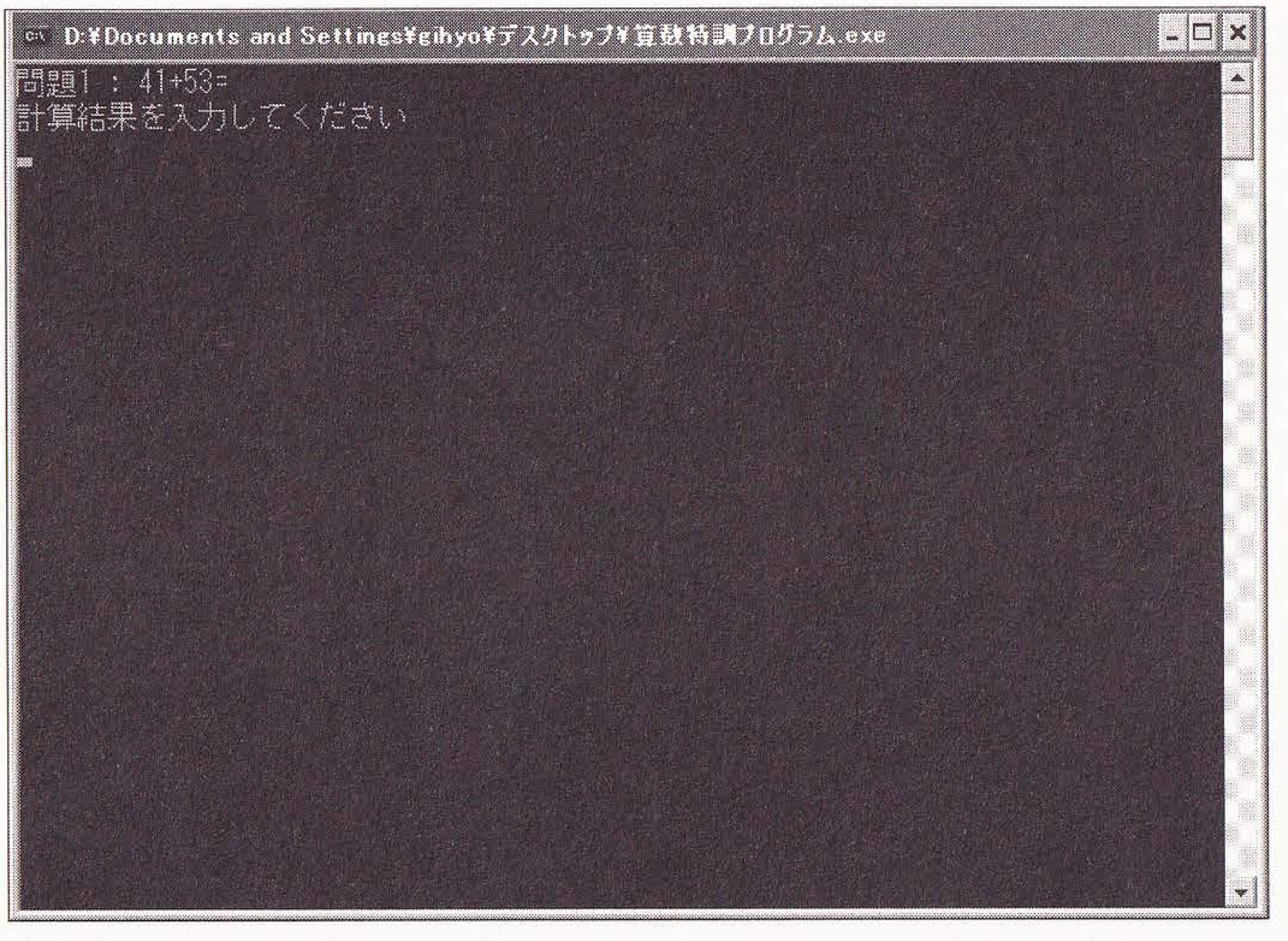
「ああ、これね、この形の変わっているアイコンだね。」

それからこのファイルをデスクトップにコピーしてみよう。



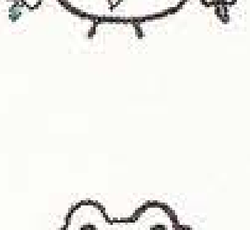
デスクトップにコピー

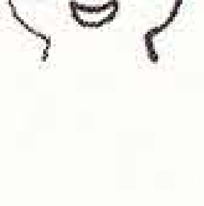
この実行ファイルがあればプログラムは動くことができる。
それからデスクトップにコピーした[**実行ファイル**]をダブルクリックしてみよう。
すると、プログラムが動き始めるんだ。
つまりWindows OSがユーザの操作を認識することで、
[実行ファイル]のアイコンが選択され、
そのファイルを起動する指示を受けたと認識するわけだ。



プログラムの起動と問題の表示

さて、プログラムは動いたかな。


「本当にプログラムが動いた。」

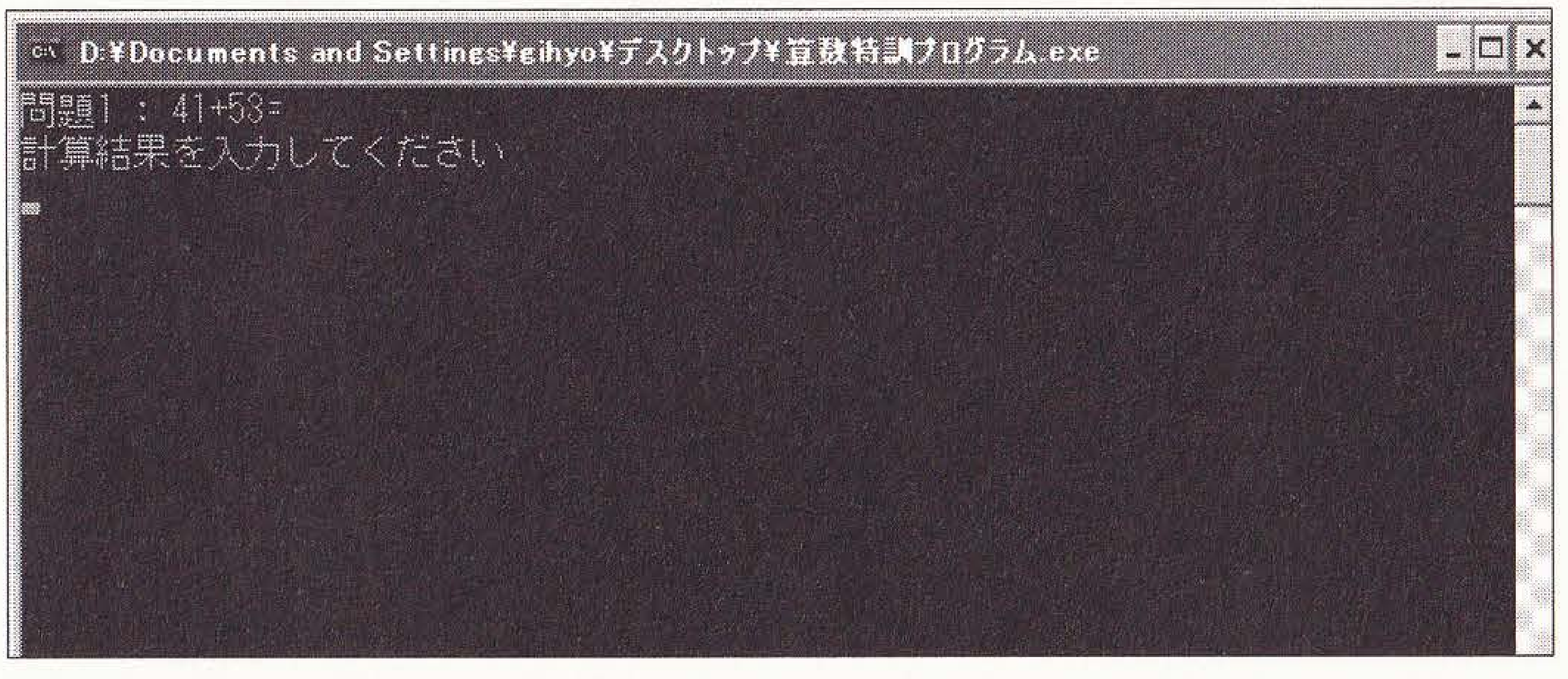

「早速、問題が表示されて出てきたよ。」

データのやりとり

このプログラムを実行すると、ソースプログラムの問題を表示する部分まで進んだことになる。つまり特訓プログラムの24行目まで一気に動いてしまうんだ。コードでいうと、問題表示を表わしている関数18行～24行目のprintf関数だね。

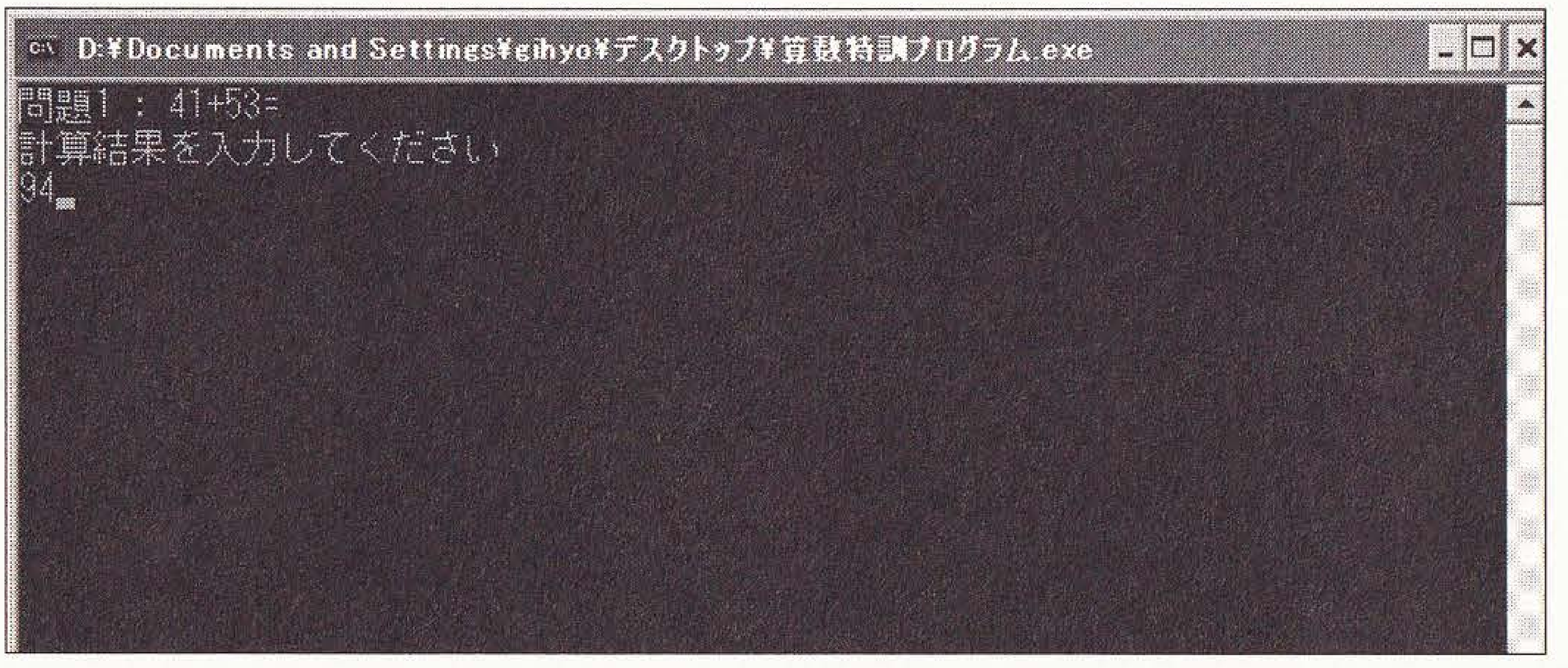
```
18        printf("問題");      //問題を表示する
19        printf("%d : ", bango);
20        printf("%d", saisyonosuuji);
21        printf("+");
22        printf("%d", nibannmenosuuji);
23        printf("= ¥n");
24        printf("計算結果を入力してください¥n");
```

この関数は、標準出力であるモニターにプログラムの中に書いてある文字列のデータを表示するように指示しているわけだ。
そして、問題の表示後は、くま君のキー入力待ちの状態になる。
プログラムの起動後と特に変わらないけれど、入力を待っている状態にあるんだ。



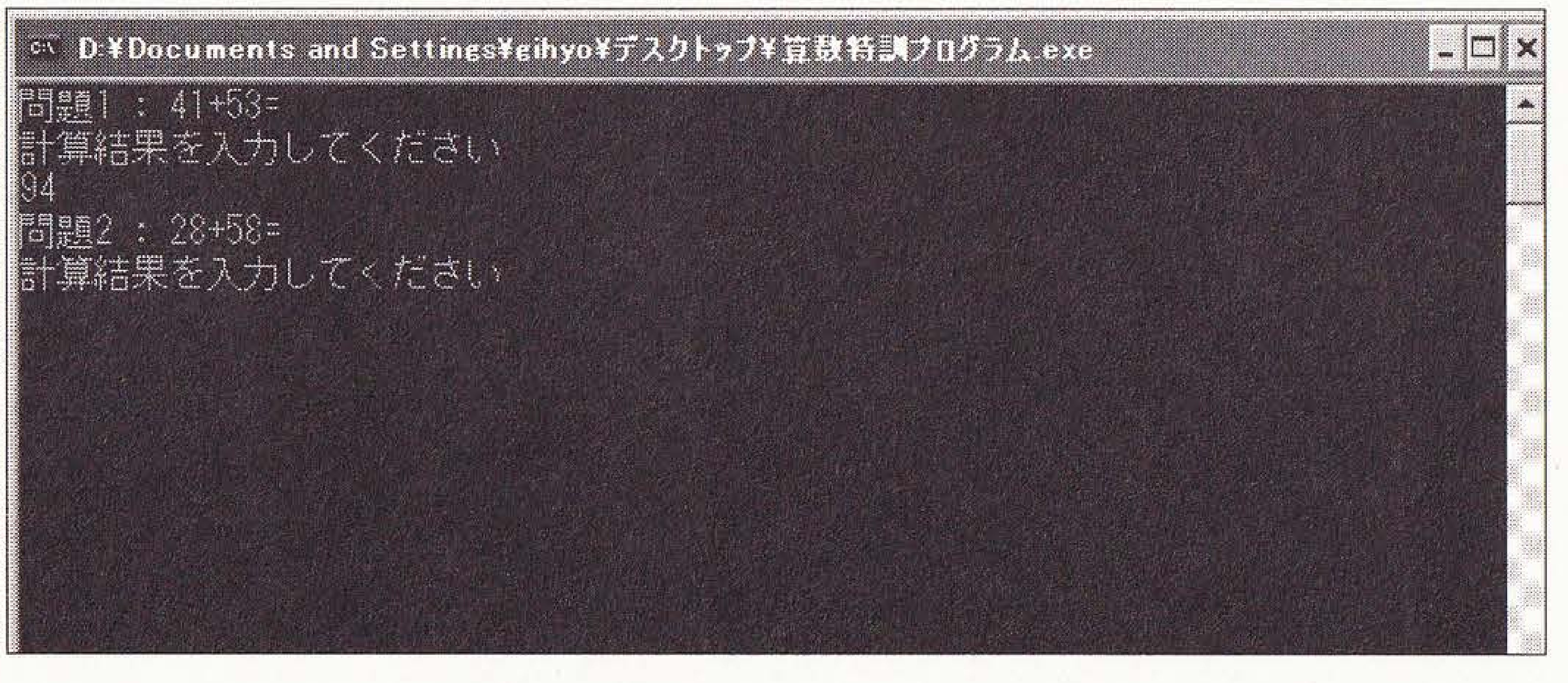
キー入力待ち

さて、計算結果の答え「94」を打ち込んでみよう。
数字の「94」を打ち込むと、そのまま「94」を表示する。
この段階では、まだプログラムにデータは伝わっていない。



計算結果の入力

次にEnterキーを押してみよう。こうすることでキー入力が確定して、キー入力したデータがプログラムに伝わるんだ。



入力の確定

つまり、Enterキーを押すことで、scanf関数がキー入力された数値を引数で指定した変数（"%d ,&, suuji"）にコピーすることを意味している。ここで変数suujiにくま君がキー入力した数値が入るんだ。

```
25        scanf("%d",&suuji);
          //入力された数を変数suujiに取得する
```

このように、パソコンはプログラムが動き始めると、
プログラムにしたがって、
必要なメモリや入出力装置を関連づけて処理を進めていくんだ。

printf関数、scanf関数は、キー入力やディスプレイ表示をするために、プログラムと標準入出力デバイスの間でデータのやりとりをしているということがわかる。

一方、プログラムの中では、変数や定数を使って、データのやりとりをしているんだ。

たとえば、48行目の呼び出される関数int ransuu(void)（P227参照）では、14行目の呼び出し元のmain関数saisyonosuuji = ransuu( );に61行目のreturn(suuji);の戻り値を戻すことで、データをやりとりしている（15行目のnibanmenosuuji = ransuu( );の数字も同様）。

呼び出される方の関数にこのように書くことで、呼び出した側にデータを渡すことができるんだ。

● プログラム中の判断と結果の表示

ここでは、データのやりとり後のプログラム中の判断と
結果の表示について説明するよ。

25行目のscanf関数でキー入力を受け付けたら、
次は、26行目の処理だ。

```
26        if(kotae == suuji){
```

ここでは、if文に続く（ ）内の条件式の結果によって、プログラムの動き方を変える。

26行目のif文の条件式は「kotae == suuji」で、変数同士の内容を比較していることを意味し、変数suujiはくま君がキー入力し「Enter」キーで確定した数値が入る。変数kotaeと変数suujiが同じなら、くま君は正しい答えを出せたということだね。

つまり、パソコンはここで、くま君の問題を解いた後のキー入力が正しいかどうかの判定をしているということになる。

くま君の解答が正しい場合には、
次のコードの、27行目を実行する。

```
27        break;
```

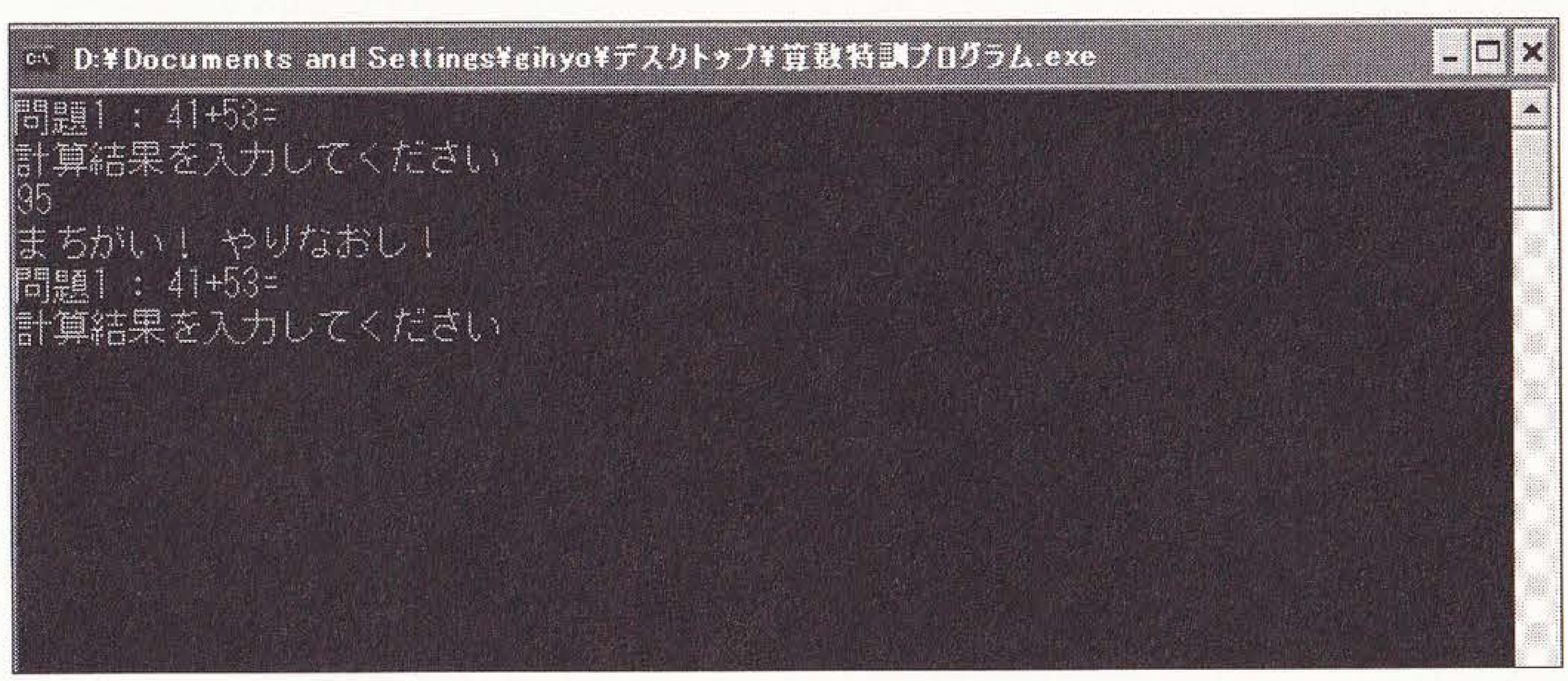
このbreak文は、今処理している17行目からのwhile文をbreakするということを表わし、while文を抜けて、13行目のfor文の繰り返し処理の初めに戻るということを示しているんだ。つまり、次の問題に進むことになる。
一方、くま君の解答が間違いだった場合には、
条件は成立しないので、

```
29          else  {
```

上記のコードに移り、

```
30                  printf("まちがい！　やりなおし！¥n");
                            //不正解の場合は同じ問題を繰り返す
31              }
32          }
33      }
```

30行目のprintf関数では、「まちがい！　やりなおし！」が表示され、間違えた問題が再度表示されることになるんだ。



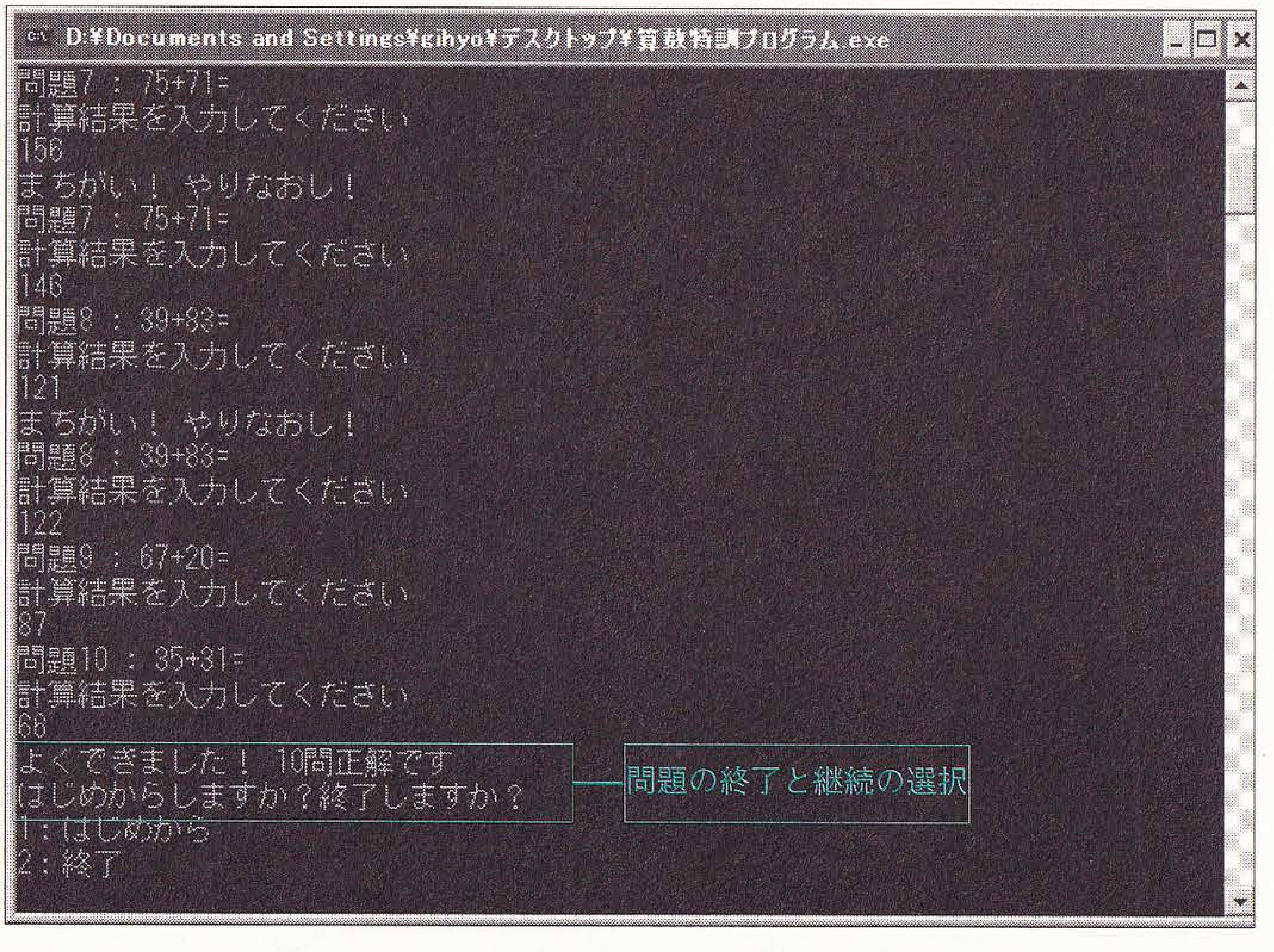
● 間違いの表示

ここではbreak文が出てこないので、この場合には、

```
while(1){
```

に戻り、繰り返して同じ問題を表示する、ことを表わしているんだ。そして、問題は10問正解になるまで、繰り返し表示されていくんだ。

最後に、くま君が10問の問題に正解した時は、変数bangoが10になり、繰り返しを終了する。



問題の終了

プログラムの終了

プログラムの終了の画面通り、くま君が算数の問題10問をクリアした後は、このfor文を終了して、次の文に進むんだ。

```
34 while(1){
```

ここでは、10問の問題をもう一度やるかどうかを確認する。
34行目～38行目のprintf関数では、10問の問題を終了したので、ねぎらいの言葉の、

```
35         printf("よくできました！10問正解です¥n");
```

「よくできました！　10問正解です」
を表示するようにしている。
その後、続けて特訓を続けるか否かの確認を表示する。

```
36         printf("はじめからしますか？終了しますか？¥n");
```

その次のscanf関数は、問題の解答の時と同じように、キー入力（1:はじめから）または（2：終了）を変数owariに入れているね。

```
39         scanf("%d",&owari);
           //選択された番号を変数owariに取得する
```

この変数owariの判断文を見ておこう。

```
40         if( owari == 1 ){
```

```
            goto START;   //owariが1の場合はSTARTラベルの位置にもどって特訓の再開
        }
        else if( owari == 2 ){
            break;   //owariが2の場合はプログラムを終了する
        }
```

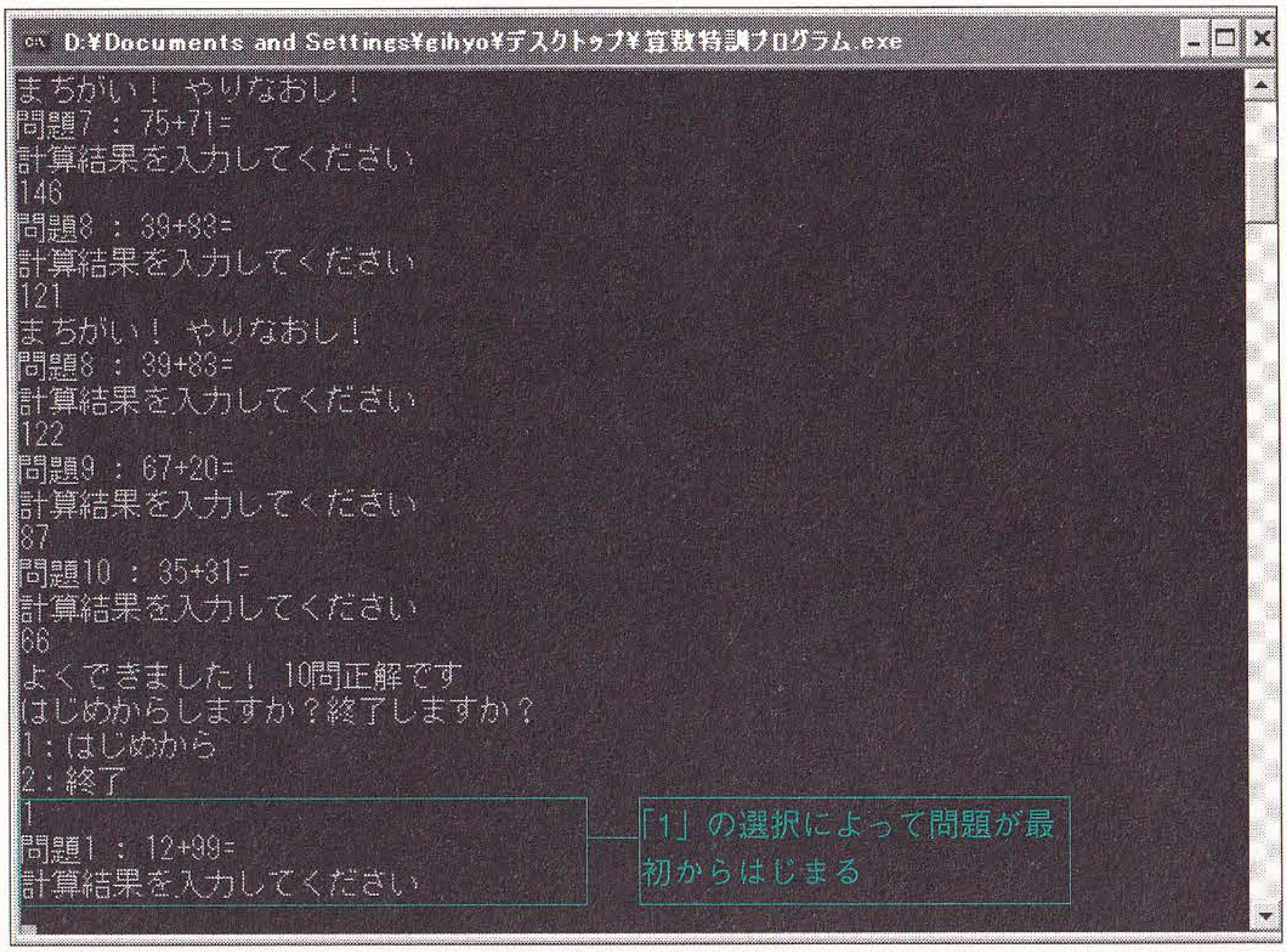
変数owariの値は、printf関数で表示した選択肢
「1：はじめから」または「2：終了」から選ばれた値で、owari = 1（1：はじめから）なら特訓を継続することを意味する。
この時、

```
        goto START;
```

の指示によって、

```
        START:   //はじめから開始する時のラベル
```

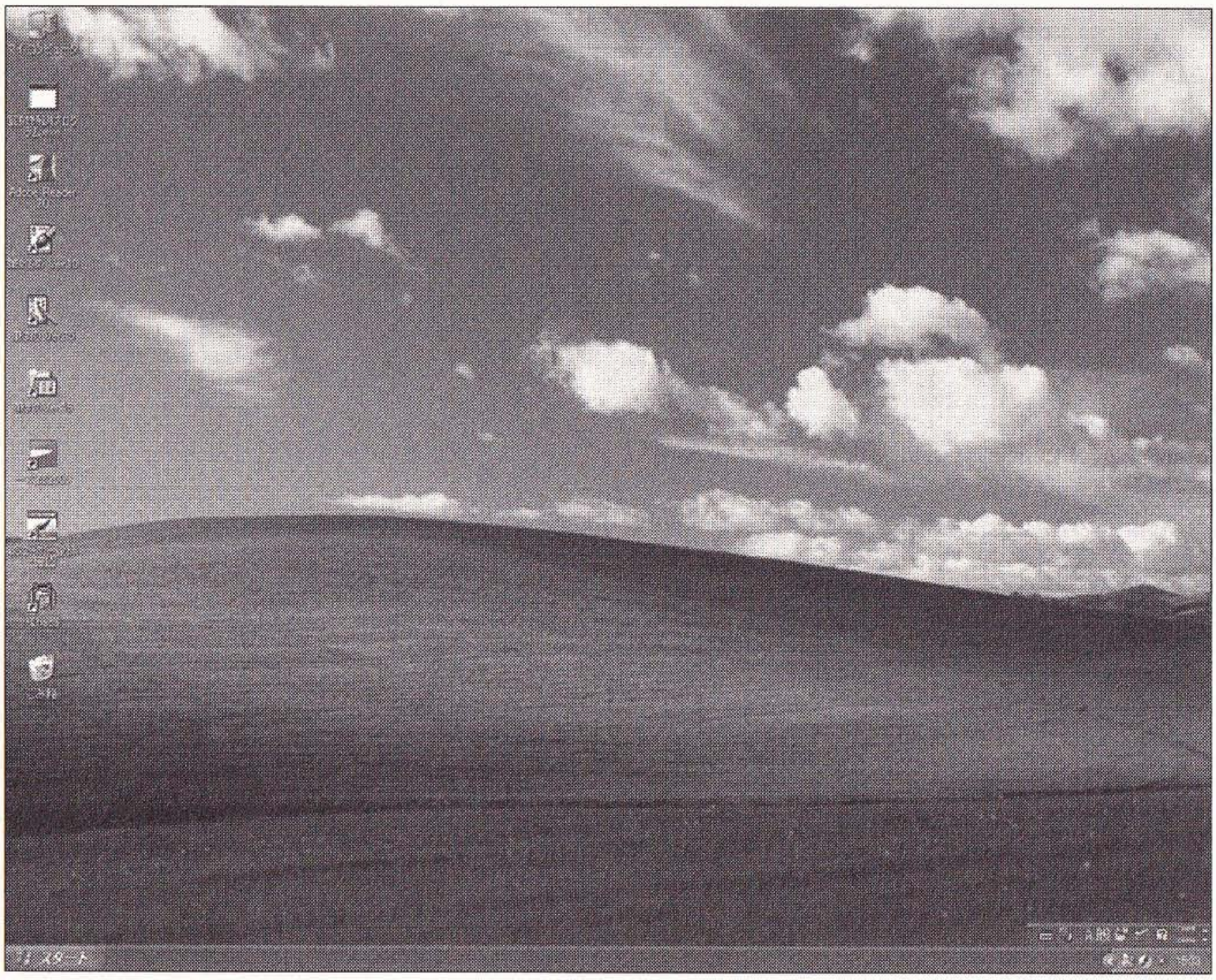
へ移り、最初からプログラムを繰り返す、ということになる。



● 問題の開始

一方、「2：終了」ならプログラムが終了し、デスクトップ画面が表示されるね。

```
else if( owari == 2 ){
break; //owariが2の場合はプログラムを終了する
}
```



● プログラムの終了

このwhile文では、ここから抜けると他に処理するところがなくなり、main関数が終了する。
main関数が終了するということは、このプログラムが終了するということを表す。このプログラムをおしまいにすると、パソコンでプログラムのために動いていたウインドウ（DOS窓）が閉じて、確かにプログラムが終了したということがわかるね。



「本当だ、プログラムが閉じちゃったね。」

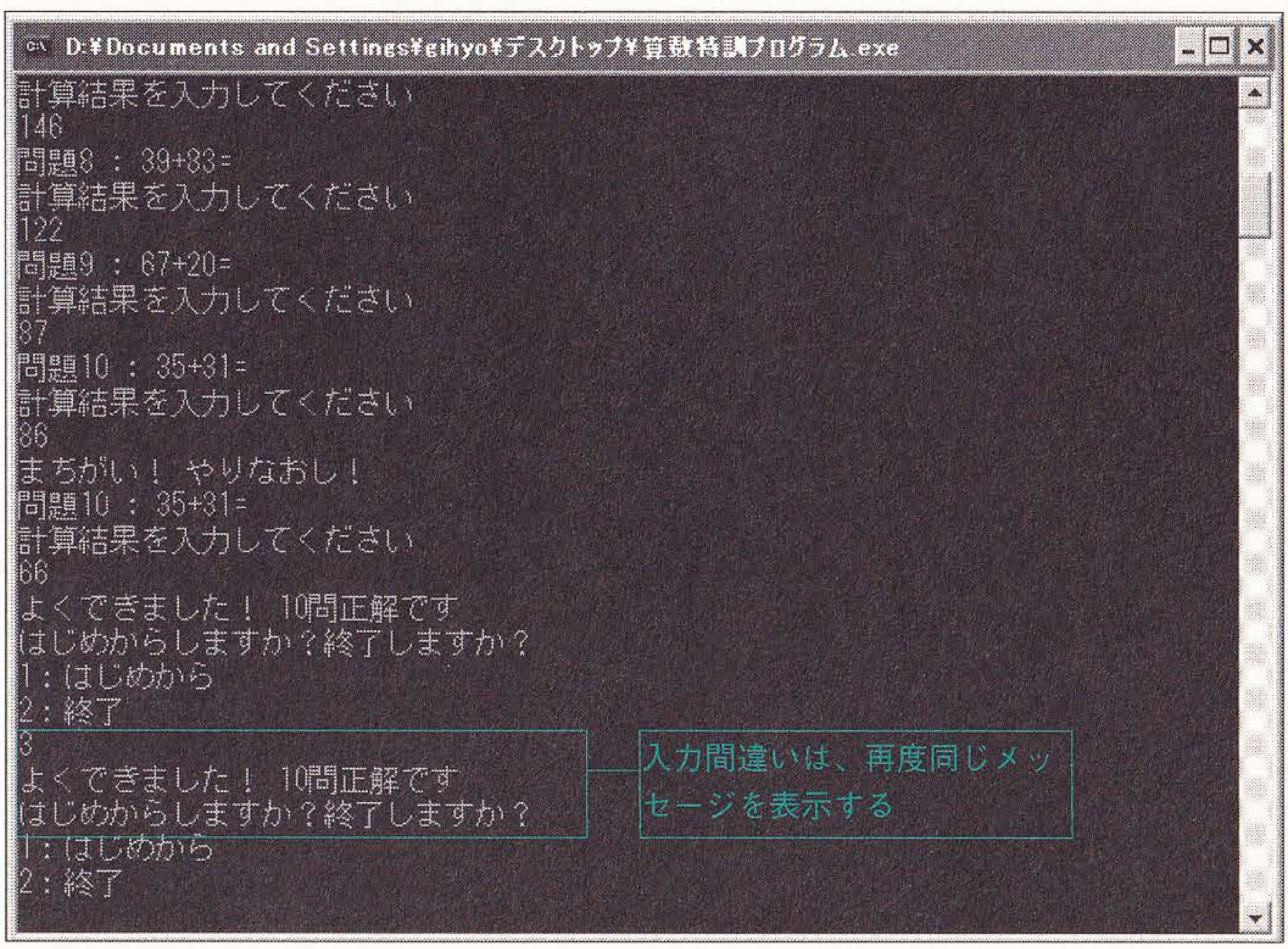


「ゲームみたいで面白いね。またこのソフトを使いたくなってきた。」



「お父さん、最後にキー入力を間違えたらどうするの？」

いいところに気がついた。
最後にここで注意しなければならないのは、
それ以外の数値をキー入力した場合だ。
“1” と “2” 以外の数字（たとえば、3とか7）をキー入力した場合には、もう一度、よくできました！ 10問正解です。
はじめからしますか？　終了しますか？
と同じメッセージを再度表示することにし、再度キー入力を促すんだ。



● キー入力間違い

06
まとめ

これまで、パソコンの電源を入れてからのコンピュータの動きやその内部でのプログラムデータのやりとりについて解説してきた。
人間の命令は、コンピュータがわかるようにプログラム（データ）にされ、各装置を通して、処理を実現している、ということが理解できたかな？
実際のプログラムでは、関数、演算子、制御文などを使って、効率よく処理が行われているんだね。
最後にこれまで学んできたことをまとめてみよう。

レッスンの総まとめ

- コンピュータは入力装置、演算装置、記憶装置、出力装置で構成されている
- コンピュータはプログラムの指示によって動く。その指示は、ONとOFFの電気信号（データ）である
- プログラミングとは、人間のして欲しい指示から、プログラムを作成・実行・テストするまでの作業のことをいう
- プログラミングは、分析、設計、実装、検査の各作業を通して行われる
- データは、メモリ上の変数に一時記憶され、必要なときに読み出されるこのとき、アドレスは変数データの場所を特定する働きをする
- プログラムは、すべて関数の集まりでできている。
- 関数は、プログラムに記述すべき処理をmain関数とそれ以外の関数に分けて、必要なときにmain関数から別の関数を呼び出して処理を行っている
- プログラムの処理は、プログラムと標準入出力との間で、メモリや入出力の関連付けをすることで行われる。また、プログラムの内部では、変数や定数を使って、データをやりとりして処理をする



ことのおわり、それから･･･

森の算数選手権で、くまは惜しくも2位に終わった。
くまは、今年こそは、と自ら算数特訓プログラムに改良に改良を重ね、
掛け算や割り算をはじめ、
あらゆる算数の練習をできるようにしてしまった。
その成果でなんと、次の年にはついに優勝を果たしたのだった。



数年後、くまは、森の算数選手権のことはすっかり忘れて、コンピュータ学習ソフトの会社を作ってしまった。
今でも、コンピュータやプログラミングが面白くて仕方がないらしい。
最近では、くま語を日本語に翻訳するプログラムを作り始めた。それにしても、なんと前向きなくまだろう･･･。
人間たちも見習いたいものだ。



さくいん

● ま行

● や行

● ら行

● わ行

著者略歴

杉浦英樹（すぎうらひでき）

1986年に富士ゼロックス株式会社に入社。以来、複写機の制御ソフト開発に携わる。ソフトウェアはわかりやすく、シンプルにつくることが最優先との信念のもと、日夜、わかりやすく確実なソフトのつくり方を追求している。

装丁 平塚光明（パイデザ）
カバー／本文イラスト 西山亜紀
本文デザイン／レイアウト デジカル デザイン室
編集 木田朱美

本書へのご質問・ご意見・ご感想はFAXまたは書面にて、以下の住所でお受け付けしております。電話でのお問い合わせには一切お答えいたしかねますので、あらかじめご了承ください。尚、お問い合わせの際にお知らせいただいた個人情報は返答以外の目的には使用いたしません。また、返答後は速やかに削除させていただきます。

〒141-8676
東京都品川区上大崎3-1-1
株式会社技術評論社　書籍編集部
「自分のペースでゆったり学ぶ
Cプログラミング超入門」係
FAX番号　03-5745-7831

自分のペースでゆったり学ぶ Cプログラミング超入門

平成18年6月25日初版　第1刷発行

著者　杉浦　英樹
発行者　片岡　巌
発行所　株式会社技術評論社
東京都品川区上大崎3-1-1
電話　03-5745-7800　販売促進部
03-5745-7830　書籍編集部
印刷／製本　日経印刷株式会社

定価はカバーに表示してあります。

ISBN4-7741-2785-X C3055
Printed in Japan



自分のペースでゆったり学ぶ　Cプログラミング超入門

ISBN4-7741-2785-X C3055 ¥1880E

定価（本体1880円＋税） M-code 414931

9784774127859

1923055018808

